The AHA Model

Revision: 12809

Generated by Doxygen 1.9.1

Chapter 1

The AHA Model: Evolution of decision making and behaviour

1.1 Overview of The AHA Model

This is a large scale simulation model (under development) that implements a general **decision-making architecture** in **evolutionary agents**. Each agent is programmed as a whole virtual organism including the genome, rudimentary physiology, the hormonal system, a cognitive architecture and behavioural repertoire. They "live" in a stochastic spatially explicit virtual environment with physical gradients, predators and prey. The primary aim of the whole modelling machinery is to understand the evolution of decision making mechanisms, personality, emotion and behavioural plasticity within a realistic ecological framework. An object-oriented approach coupled with a highly modular design not only allows to cope with increasing layers of complexity inherent in such a model system but also provides a framework for the system generalizability to a wide variety of systems. We also use a "physical-machine-like" implementation philosophy and a coding standard integrating the source code with parallel detailed documentation that increases understandability, replicability and reusability of this model system.

The cognitive architecture of the organism is based on a set of motivational (emotional) systems that serves as a common currency for decision making. Then, the decision making is based on **predictive assessment** of external and internal stimuli as well as the agent's own motivational (emotional) state. The agent makes a subjective assessment and selects, from the available repertoire, the behaviour that would reduce the expected motivational (emotional) arousal. Thus, decision making is based on predicting one's own internal state. As such, the decision-making architecture integrates motivation, emotion, and a very simplistic model of consciousness.

The **purpose** of the AHA model is to investigate a general framework for modelling proximate decision-making and behavior. From this we will investigate adaptive goal-directed behaviour that is both guided by the external environment and still is endogeneously generated.

Other research topics include individual differences, personality as well as consequences of emotion and personality to population ecology.

We think that understanding and modelling complex adaptive behaviour requires both extraneous (environmental) factors and stimuli as well as endogeneous mechanisms that produce the behaviour. Explicit proximate representation of the motivation and emotion systems, self-prediction can be an important component in linking environment, genes, physiology, behavior, personality and consciousness.

- The AHA! Project Development Pages: `http://ahamodel.uib.no`
- Building blocks of the AHA Model: Building blocks of the AHA model
- The Cognitive Architecture of the agent: The Cognitive Architecture

Fortran is used due to its simplicity and efficiency. For example, check out this paper: `Why physicists still use Fortran?`.

Main features of modern Fortran that are used in the AHA model code are briefly outlined in the `README_NF` .

1.2 Version information

This is the model version information parsed from the main Subversion repository https://svn.uib.no/aha-fortran or Bitbucket Mercurial-based repository https://bitbucket.org/ahaproject/hedg2_01 (the latter is currently used just as a mirror of the Subversion branch).

```
$Id: m_common.f90 9552 2020-06-06 07:26:50Z sbu062 $
```

Version information is also saved as two variables that can be passed to the logger (see commondata::logger_init()) and outputs and file names:

- commondata::svn_version_string, full version string as above;
- commondata::svn_version, version number (hex id).

1.3 Working with the Model

A "Getting Started" introduction is available in the README.md file.

1.3.1 Overview of the AHA Fortran modules

The Modelling framework is composed of two separate components:

- HEDTOOLS (also mirror at bitbucket), modelling utilities and tools implemented as portable Fortran modules, not object-oriented, that have general applicability and are used for data conversion, output, random number generation and execution logging. HEDTOOLS modules are designed such that they can be used in many different simulation projects, not only the AHA model;
- The AHA model, an object oriented evolutionary agents simulation framework implementing reusable module components.

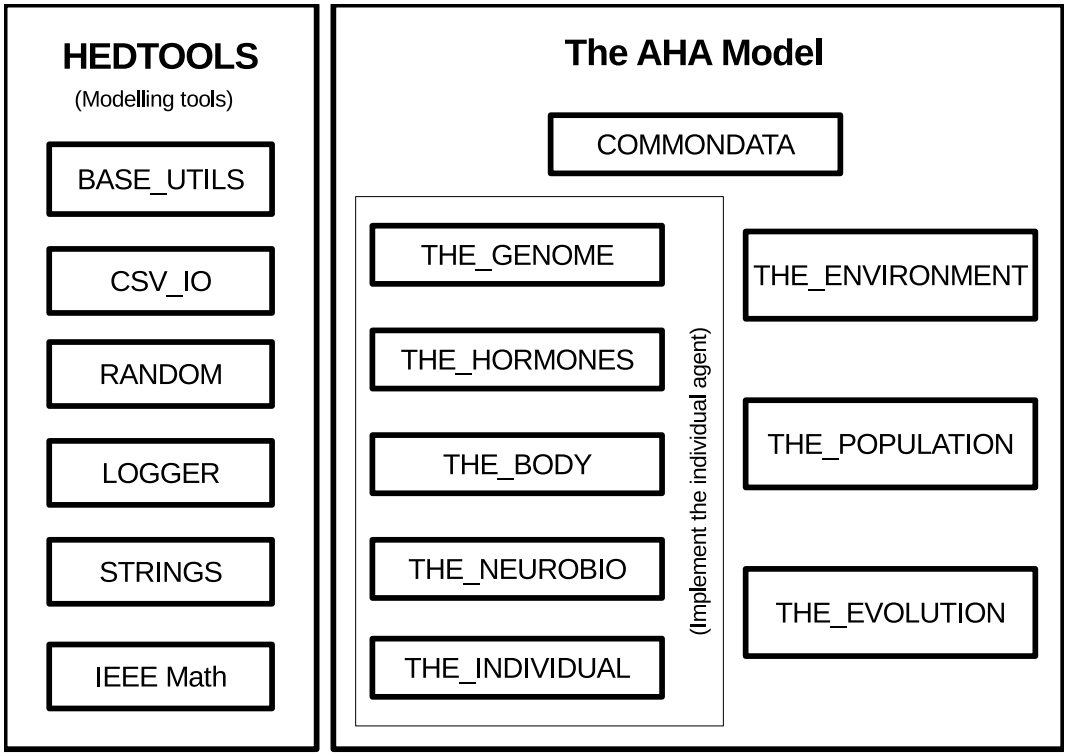


Figure 1.1 Overview of AHA Fortran modules

1.3.2 Running the model

Building and running the mode is based on the GNU Make system.

- To **build** the model from the source code issue:
  `make`
- To build and **run** the model issue this command:
  `make run`
- **Delete** all build-related, data and temporary files:
  `make distclean`
- Get a quick **help** from the make system:
  `make help`
- The compiler can be provided via `FC` variable, e.g.
  `make run FC=ifort`
- Building with debug symbols is controlled by setting the `DEBUG` variable.
  `make run FC=ifort DEBUG=1`

See Makefile for build configuration. To get more information on GNU Make see `AHA Modelling Tools Manual`, `Using Microsoft Visual Studio` and `Using Code::Blocks IDE`.

1.3.3 Environment variables

The model makes use of several environment variables to control certain global aspects of the execution. These variables have the `AHA_` prefix and are listed below.

- `AHA_DEBUG=TRUE` sets the debug mode;
- `AHA_SCREEN=YES` sets logger to write to the standard output in addition to the log file;
- `AHA_DEBUG_PLOTS=YES` enables generation of debug plots;
- `AHA_ZIP_FILES=YES` enables background compression of big output data files;
- `AHA_CHECK_EXTERNALS=NO` disables checking for external executable modules (this is a workaround against Intel Fortran compiler bug).

They are described in details in the `commondata::system_init()` section. **Setting** the environment variable on different platforms: **Linux/Unix:**
`export AHA_DEBUG=1`
Windows:
`set AHA_DEBUG=1`
Checking if the environment variable is set and what is it value On Unix/Linux:
`echo $AHA_DEBUG`
On Windows
`echo %AHA_DEBUG%`
To check if any of the above environment variables are set is easy on Linux:
`env | grep AHA_`

1.3.4 The DEBUG mode

The protected global variable `IS_DEBUG` (commondata::is_debug) sets up the **debug mode** of execution. The debug mode results in huge amount of output and logs that significantly slows down execution. Debug mode can be set using the environment variable `AHA_DEBUG=1`, `AHA_DEBUG=YES` or `AHA_DEBUG=TRUE`. Debug mode can also be set by setting the runtime **command line** parameter to `DEBUG`, `DEBUG=1`, `DEBUG=YES` or `DEBUG=TRUE` to the model executable, e.g.
`./MODEL.exe DEBUG`
This runtime command line parameter is automatically set if the model is started using the DEBUG variable for the GNU Make:
`make run DEBUG=1`
Build Note also that `DEBUG` with make build command
`make DEBUG=1`
will build the executable for the model in the debug mode also. All compiler optimisations are turned off (`-O0`), debugger symbols (`-g`) and tracebacks are enabled, various compiler error checking options and warnings are also enabled, along with extended runtime checks.

Notably, the Intel Fortran compiler enables initialisation of variables with the signalling NaNs (`-init=snan`). Any operations on non-initialised variable(s) would then result in the runtime crash with traceback, making it easier to catch non-initialised variables.
Normal build without setting `DEBUG`, in contrast, enables various compiler optimisations making the executable run faster.

```
make
```

Notably, with the Intel Fortran compiler, the model is built with automatic parallelisation (`-parallel`), so whole array operations, `do concurrent` and `pure` and `elemental` functions are run in the `multi-threaded mode` whenever the compiler is "sure" that these operations can be safely performed. See the Makefile for specific compiler and debugging options and the `GNU make` for a general overview of the system. A overview of Intel Fortran parallel processing options can be found `here` and `here`.
Combining the shell environment variable `AHA_DEBUG` and the `DEBUG` make variable allows to control how the model executable is build and run. For example, *building* it in *non-debug* mode, i.e. with all the optimisations for fast (and possibly multi-threaded) execution:

```
make
```

can be combined with *running* the model in the *debug* mode:

```
export AHA_DEBUG=YES
make run
```

In such a case, the model is built and executed in the "fast" optimised and possibly multi-threaded mode, but still prints all the numerous debugging outputs into the logger.
The system initialisation procedure commondata::system_init() and commondata::logger_init() provide more details on initialisation and logging.

1.3.5 The lock file

At the start of the simulation, the program creates an empty **lock file** defined by commondata::lock_file. The lock file is created at the end of commondata::system_init() and and is deleted by the procedure commondata::system_halt().
The lock file indicates that the simulation is (still) running and could be used by various scripts, cron jobs etc to make sure the simulation is finished. For example, a script can check from time to time if the lock file exists and if not, issue a command to zip all data outputs and transfer them to a remote analysis server.

1.3.6 The stop file

The stop file is an effective method to terminate the simulation at any generation and avoiding any destructive consequences that would be caused, for example, by using the `Ctrl+C` keypress or `kill` command. Thus, the stop file is a signalling file.
The name of the stop file is defined by the parameter commondata::stop_file. This file is checked upon each new generation of the Genetic Algorithm in the_evolution::generations_loop_ga() (`CHECK_STOP_FILE` block). If this file is found, simulation will not proceed to the next generation and will just stop as if it were the last generation. All the data are saved as normal.

Note

The stop file can be easily created on Lunux terminal by this command: `touch stop_simulation_running.lock`. On Windows, these two command can be used for the same purpose: `type nul > stop_simulation_running.lock` or `copy NUL stop_simulation_running.lock`. The lock file is deleted by `make cleandata` command. Note also that once the stop file is created in the model running directory, actual stop is not immediate: it will be executed only at the next generation, the currently running generation is completed to its end as normal.

1.3.7 Output numerical data

Data could be output from the model. The standard format for data output is `CSV`. Its advantage is that it is based on plain text and is human readable nut can be easily imported into spreadsheets and statistical packages. The most straightforward use for CSV is for vector-based data and two-dimensional matrices, including sets of vectors (e.g. 'variables'/columns with 'observations'/rows).
CSV output is based on the `CSV_IO` module in `HEDTOOLS`. There is also an object oriented wrapper for CSV_IO: file_io.
The model also includes a few standardised CSV output procedures for saving various characteristics for a whole population of agents:

- the_population::population::save_csv() – save various condition data;
- the_population::population::save_genomes_csv() – save the complete genome data of the agents;
- the_population::population::save_memory_csv() – save the perceptual and emotional memory stacks of the agents;
- the_population::population::save_movements_csv() – save the latest movement history of the agents.
- the_population::population::save_behaviour_csv() – save the behaviour history of the agents: successive labels of behaviours executed.

Additionally, the genetic algorithm subroutine the_evolution::generations_loop_ga() implements a sub-procedure

- generation_stats_record_write() – produces various generation-wise statistics, such as the number of surviving agents, average body mass etc.

The following procedures output the properties of the the_environment at a particular time step:

- the_environment::food_resource::save_csv() – saves all the (stochastic) food items that compose the the_environment::food_resource.
- the_environment::habitat::save_predators_csv –saves all the (possibly stochastic) predators that are implemented within the habitat.

1.3.8 The Model descriptors

The model includes several descriptors that can be used for descriptive notes on the model.

- commondata::model_name – model name: a short name of the model;
- commondata::model_descr – model description: a longer, one line description of the model;
- The Model Abstract – A brief text that can span several lines of text and is kept in a separate file with the name defined by the commondata::model_abstract_file parameter. If this file is absent, model description is used.

These descriptors can appear in the model logger output and form parts of the output data files.
In particular, because the Model Abstract is stored in a separate text file, it can contain dynamically updated information, such as the latest version control message log output. Subversion and Mercurial commit hooks can be used to implement such a functionality.
To append version information (Subversion) to the Model Abstract file manually do this:

```
svn log -l1 » abstract.txt
```

1.3.9 Coding and documenting style

Using a consistent coding style increases the readability and understandability of the program. It is also easier to search and locate specific parts. For example, using specific rules for version control commit messages make it easier to find specific changes by using regular expression syntax. The coding rules for the AHA model are relatively light.

1.3.9.1 Fortran coding style

- Clean, readable and understandable code is much better than "tricky" but more computationally efficient code.
- Obsolete, outdated and non-standard (e.g. vendor extensions) features of the Fortran language should *never* be used.
- Very modern Fortran features (e.g. many F2008 and F2015) not well supported by the available compilers should be avoided whenever possible.
- Portability is crucial: the model should be easy to build and run using different compilers, GNU free software *gfortran* is the compiler of priority.
- Frequent checking for errors (and even *possible* errors) is important; correct computation has much higher priority than just speed.

- Use generic programming techniques (e.g. optional arguments, generic interfaces) for higher extensibility.
- Always use explicit `intent` declared for all subroutines and functions.
- The line length should not exceed 80 characters; use continuation lines and indents to make the structure clear.
- Spaces (not tabs) are used for indentation, standard indent is 2 characters.
- Use *lowercase* names for all local and Fortran intrinsic objects.
- Global variables (not constants) should be in "CamelCase".
- Public or local *constants* (that are not changed) are in UPPERCASE.
- External or library objects, e.g. those from the HEDTOOLS are in UPPERCASE
- Module names are UPPERCASE, class names are UPPERCASE.
- Spatial objects (`SPATIAL` and `SPATIAL_MOVING`) use `position` to *set* spatial position and `location` to *get* it.
- `create` method is used to create and initialise an empty object (e.g. `SPATIAL` with missing coordinates), should not take other parameters and must be elemental (so cannot include calls to random);
- `make` or `build` method is used to create a working object with random initialisation etc.

See the `AHA Modelling Tools Manual` for more details (`http://ahamodel.uib.no/doc/HEDTOOLS.pdf`).
The `README_NF` provides a brief outline of the main features of modern Fortran (F2003/2008) that are used in the model code.

1.3.9.2 Version control style

- Commit messages noting specific subroutines or functions should include the names of these procedures in parentheses:
```
svn ci -m "hope function (hope) now accepts arbitrary raw grid arrays"
```
- Commit messages chenging only the Doxygen comments should start with 'doc:', e.g.
```
svn ci -m "doc: still notes render poorly, try top put to return"
```

1.3.9.3 Doxygen self-document style

- If something is changed in the *code*, the documentation comments should also be checked for *correspondence with the code*, Doxygen comments should *always* reflect the logic of the code and should be never outdated.
- Document procedure purpose and brief implementation in the over-the-procedure header comment block.
- *Local variables* are **not** in the Doxygen tags unless they are very important, then include full paragraph description.
- Porting and F2008 compatibility notes and warnings are not in the Doxygen tags unless they are very important.
- Unimportant implementation notes can be documented within the body of the procedure *without* Doxygen "`!>`" tags. So they are never parsed.
- *Important* implementation details are fully documented in Doxygen tags. So they can be rendered if `HIDE_IN_BODY_DOCS` is `NO`.
- Implementation details within the procedure body should be under the level 3 headers enclosed in `###`:
```
!> ### Implementation details ###
```
- Full Fortran variable object hierarchy for instantiated objects must be as normal text, i.e. **not enclosed in reverse single quotes** (as verbatim code), because percent sign is not escaped and parsed wrongly then:

```
this\%reprfact_decrement_testosterone
```

not

```
`this\%reprfact_decrement_testosterone`
```

or

```
this%reprfact_decrement_testosterone
```

or still

```
`this%reprfact_decrement_testosterone`
```

as the percent sign disappears in the parsed output.

- Reference to other subroutines and functions of the code must be in the C++ style, referring the class/module in **lowercase**:

```
!! values resulting from executing this behaviour (`reproduce::do_this()`
!! = the_neurobio::reproduce_do_this() method). This is repeated for
```

This makes them appear as cross-links in the parsed document.

- Documenting specific implementation procedure for in-type interface name declaration should follow this style (include link to implementation):

```
!> Calculate the Euclidean distance between two spatial objects.
!! See `the_environment::spatial_distance_3d()`
procedure, public :: distance => spatial_distance_3d
```

- Reference to a procedure that is defined within this (being documented) procedure (i.e. below `contains`) is using the hanging `::` notation:

```
!! The subjective capture probabilityis calculated by the sub-function
!! `::subjective_capture_prob()`.
```

- Documentation sections/subsections/subsubsections with tags are inserted like this:

```
!> @subsubsection aha_buildblocks_genome_genome Individual genome
```

They can be referred to in the documentation text using the tag like this:

```
!> @ref aha_buildblocks_individual "The individual agent" is also ...
```

- The @name tag defines an arbitrary **member group** of Doxygen objects. Membership range is defined by the @{ and @} tags. But this does not seem to work within `type` definitions to delimiter outer procedure interfaces.

1.3.10 Brief notes on computation

1.3.10.1 Float point computations

There are many possible quirks and caveats with the real type (float point) calculations on the computer. The rules of float point computations deviate from the basic arithmetic. The main issue is that real type numbers are represented as bits and have finite and limited precision. Furthermore, numerical precision can deteriorate due to rounding in computations.
The precision of the float point number representation in Fortran is controlled by the kind parameter (e.g. `real(kind=some_number)`).
Numerical precision modes. In the AHA Model, there are two basic numerical precision modes:

- commondata::srp, "Standard Real Precision" for real numbers that is normally used for all computations:
  `real(SRP) :: value`

Note

Note that commondata::srp precision model should be used in most cases.

- commondata::hrp, "High Real Precision", an extended precision that is used in a few cases where commondata::srp is not enough for valid computation (insufficient precision or inability to represent huge numbers):
```
real(HRP) :: value
```
Parenthetically, there is also an extended precision integer type defined by commondata::long parameter.

Constants. There is also a useful commondata::zero constant, which sets some "minimum distinguishable non-zero" value: it is the smallest real number *E* such that $1 + E > 1$.
The smallest positive real value is defined by the commondata::tiny_srp constant. It is used in the definition of the default numerical tolerance value for high precision calculations:

- commondata::tolerance_low_def_srp;
- commondata::tolerance_low_def_hrp (the same constant for the high commondata::hrp precision with commondata::tiny_hrp).

Note

Note that the commondata::tiny_srp and commondata::tiny_hrp values are much smaller than the commondata::zero parameter. Also, the default tolerance limits commondata::tolerance_low_def_srp and commondata::tolerance_low_def_hrp are also much smaller than the commondata::zero.

Because the low tolerance based on the commondata::tiny_srp and commondata::tiny_hrp may be too small and restrictive in many cases, the second set of tolerance limits for low-precision calculations is based on the commondata::zero parameter:

- commondata::tolerance_high_def_srp (commondata::srp real);
- commondata::tolerance_high_def_hrp (commondata::hrp real).

Note

These high tolerance values should be used as the standard `epsilon` in most cases.

The default commondata::srp values of these parameters calculated on an x86_64 platform under Linux are (an example only!):

```
ZERO: 1.19209290E-07
TINY_SRP: 1.17549435E-38
TOLERANCE_LOW_DEF_SRP: 5.87747175E-38
TOLERANCE_HIGH_DEF_SRP: 1.19209290E-04
```

These constants are reported at the start of the logger output.
Real type equality. One possible quirk in float point computation involves equality comparison, e.g.
```
if ( a == b) then ...
```
With real type data `a` and `b`, such a condition can lead to unexpected results due to finite precision and even tiny rounding errors: the numbers that are deemed *equal* may in fact *differ* by a tiny fraction leading to `a == b` condition being `FALSE`.
Instead of the exact comparison, one should test whether the absolute difference is smaller than than some predefined ε tolerance value (in the simplest case):

$$|a - b| < \varepsilon$$

The ε is chosen based on the nature of the data and the computational algorithm.
The AHA Model framework includes a specific function for testing *approximate equality* of two reals↩ : commondata::float_equal(). With this function, correct comparison is:
```
if ( float_equal(a, b, epsilon) ) then ...
```
There is also a user defined operator "float equality" `.feq.` that works as the commondata::float_equal() function, but uses a fixed default `epsilon` equal to the default tolerance commondata::tolerance_low_def_srp (or commondata::tolerance_low_def_hrp for high precision). Its benefit is that the usage almost coincides with the == (`.eq.`) operator usage:
```
if ( a .feq. b) then ...
```

See the backend procedures commondata::float_equal_srp_operator() and commondata::float_equal_hrp_operator() for details. Another similar operator is "approximate equality" `.approx.` has a much higher level of tolerance (larger error accepted)

```
if ( a .approx. b) then ...
```

See the backend procedures commondata::float_approx_srp_operator() and commondata::float_approx_hrp_operator() for details.
There is also a function for testing if a real value is approximately equal to zero: commondata::is_near_zero():

```
if ( is_near_zero(a) ) then      ! correct equivalent of if ( a == 0.0 )
```

1.3.10.2 Initialisation undefined constants

When a variable is created but not yet initialised, its value is "undefined". However, it can take some haphazard values depending on the compiler and the platform. There are special "initialisation" constants defined in commondata that set "missing" or "undefined" variable status: commondata::missing (real) and commondata::unknown (integer). By default, they are set to an unusual negative number -9999, so that any bugs are clearly exposed if a variable inadvertently uses such an "undefined" value.

1.3.10.3 Nonparametric functions

The model in many cases makes use of **nonparametric relationships** between parameters. This is based on the linear and non-linear interpolation procedures implemented in HEDTOOLS: `Interpolation routines`
Instead of defining specific function equation linking, say, parameters *X* and *Y*: *Y=f(X)*, the relationship is defined by an explicit **grid** of *X* and *Y* values without specifying any equation.
In the simplest two-dimensional case, such a grid is defined by two parameter arrays, for the *abscissa* and the *ordinate* of the nonparametric function. Any values of this function can then be calculated based on a nonlinear interpolation. This makes it very easy to specify various patterns of relationships even when exact function is unknown or not feasible.
An example of such a nonparametric function is the_neurobio::gos_global::gos_find().

1.3.11 Links for more information

AHA/BEAST Resources:

- AHA Model repository mirror on Bitbucket: `https://bitbucket.org/ahaproject/hedg2_01`
- Full documentation for the AHA Model in PDF: `http://ahamodel.uib.no/doxydoc/refman.pdf`
- HEDTOOLS repository mirror on Bitbucket: `https://bitbucket.org/teg_uib/hedtools`
- HEDTOOLS documentation in HTML format: `http://ahamodel.uib.no/doc`
- HEDTOOLS documentation as a single PDF file: `http://ahamodel.uib.no/doc/HEDTOOLS.pdf`
- Development statistics for the model: `http://ahamodel.uib.no/devstat/dir_branches_budaev_HEDG2_01.html`
- TEG development statistics from Subversion: `http://ahamodel.uib.no/devstat`

Tools web resources:

- GNU Fortran documentation: `https://gcc.gnu.org/onlinedocs/` (note that the model code targets features of the pre-version 5 because of their wider availability in Linux repos.)
  - GNU Fortran 4.9: `https://gcc.gnu.org/onlinedocs/gcc-4.9.4/gfortran/`
- Intel Fortran compiler documentation: `http://ahamodel.uib.no/intel/` (note that the model code targets version 17.0)
  - Automatic parallelisation is `here`
- Doxygen documentation: `http://www.doxygen.org/`

  - Special tags: http://www.stack.nl/~dimitri/doxygen/manual/commands.html
  - Formulas: http://www.stack.nl/~dimitri/doxygen/manual/formulas.html
- GNU Make: https://www.gnu.org/software/make/

1.4 Building blocks of the AHA model

1.4.1 The environment

The environment is a full 3D space, that is has the class the_environment::spatial as the elementary base primitive. The the_environment::spatial is a single object that has *X, Y and Z* (depth) coordinates. the_environment::spatial_moving extends the basic the_environment::spatial object by allowing it to move. Furthermore, the_environment::spatial_moving includes a history stack that records the latest history of such a spatial moving object. Examples of spatial moving objects can be food items (the_environment::food_item), predators (the_environment::predator), and more complex objects composed of several the_environment::spatial components like the_environment::environment.
The basic environment where the agents "live" is very simplistic in this version of the model. It is just and empty box. The box is delimited by the basic environmental container: the_environment::environment class. The the_environment::habitat class is the ecological "habitat", an extension of the basic the_environment::environment that adds various ecological characteristics and objects such as the_environment::food_resource, array of the_environment::predator objects etc.
Normally, the movement of the agent is limited to a specific the_environment::environment container with its own the_environment::habitat. All the habitats that are available for the agents are arranged into a single global public array the_environment::global_habitats_available.
The individual agent is also an extension of the the_environment::spatial class: the_environment::spatial → the_environment::spatial_moving → the_genome::individual_genome → ... → the_population::member_population.

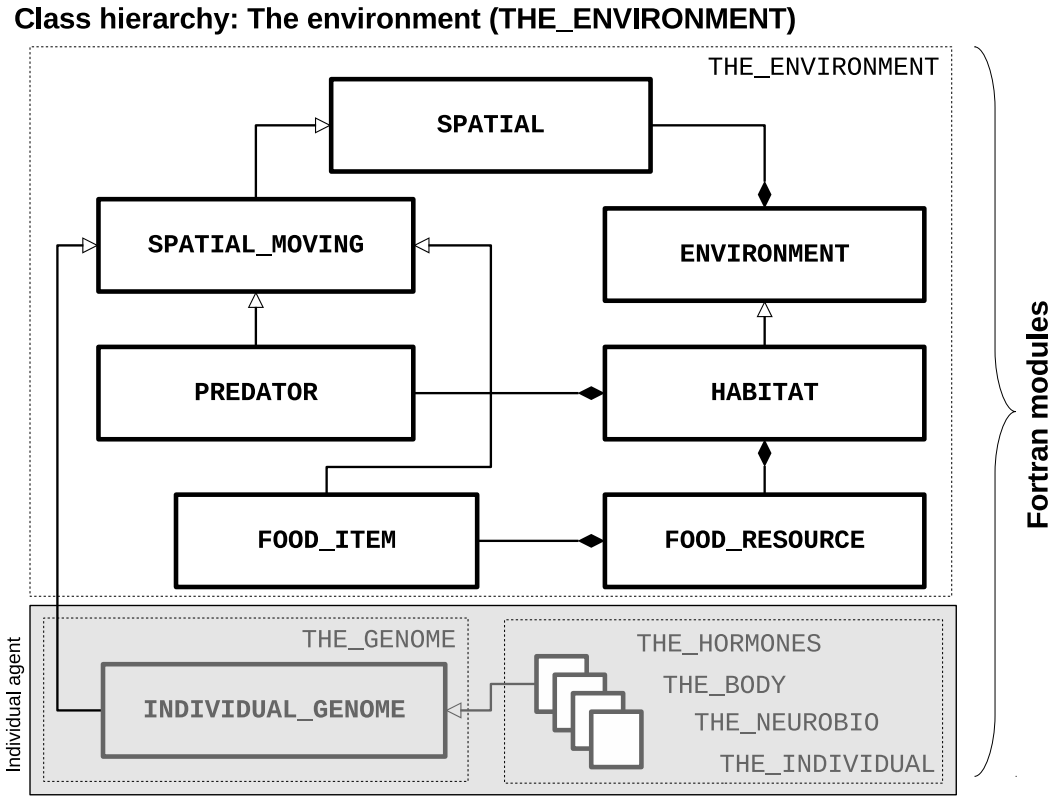


Figure 1.2 Environmental objects

1.4.2 The genome structure

A brief outline of the genetic architecture, defined in the_genome, is presented on this scheme.

- the_genome::gene
- the_genome::chromosome
- the_genome::individual_genome

1.4.2.1 Gene

The agent has genes (class the_genome::gene) that are arranged into chromosomes (class the_genome::chromosome). Each gene also includes an arbitrary number of additive components (see the_genome::gene).

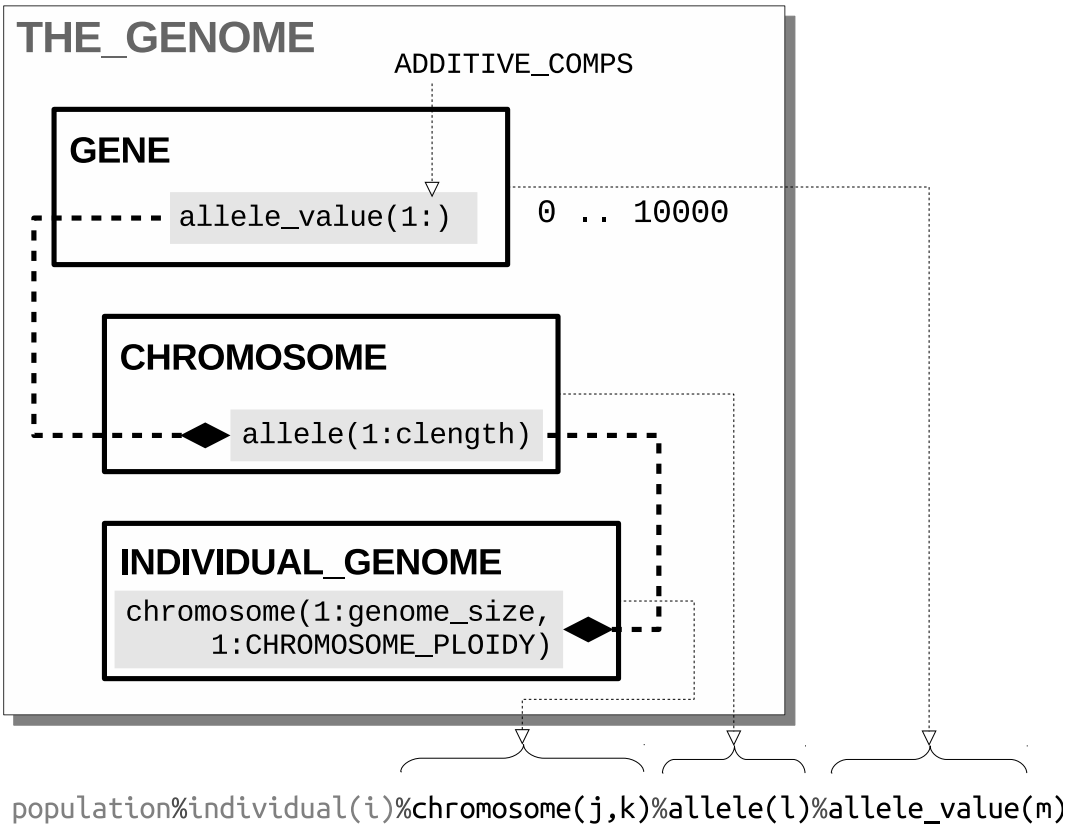


Figure 1.3 The genome structure

1.4.2.2 Chromosome

Here is a brief outline of the chromosome structure. The chromosomal architecture allows arbitrary ploidity (however haploid is not supported, although can be easily added), i.e. agents with diploid and polyploid genomes can be implemented. Ploidity is defined by a single parameter commondata::chromosome_ploidy.
Correspondence between the genotype and the phenotype (hormones, neurobiological modules etc.) is represented by boolean **Gene x Phenotype matrices**. Any arbitrary structure can be implemented, many traits controlled by a single gene, many genes controlling a specific single trait.

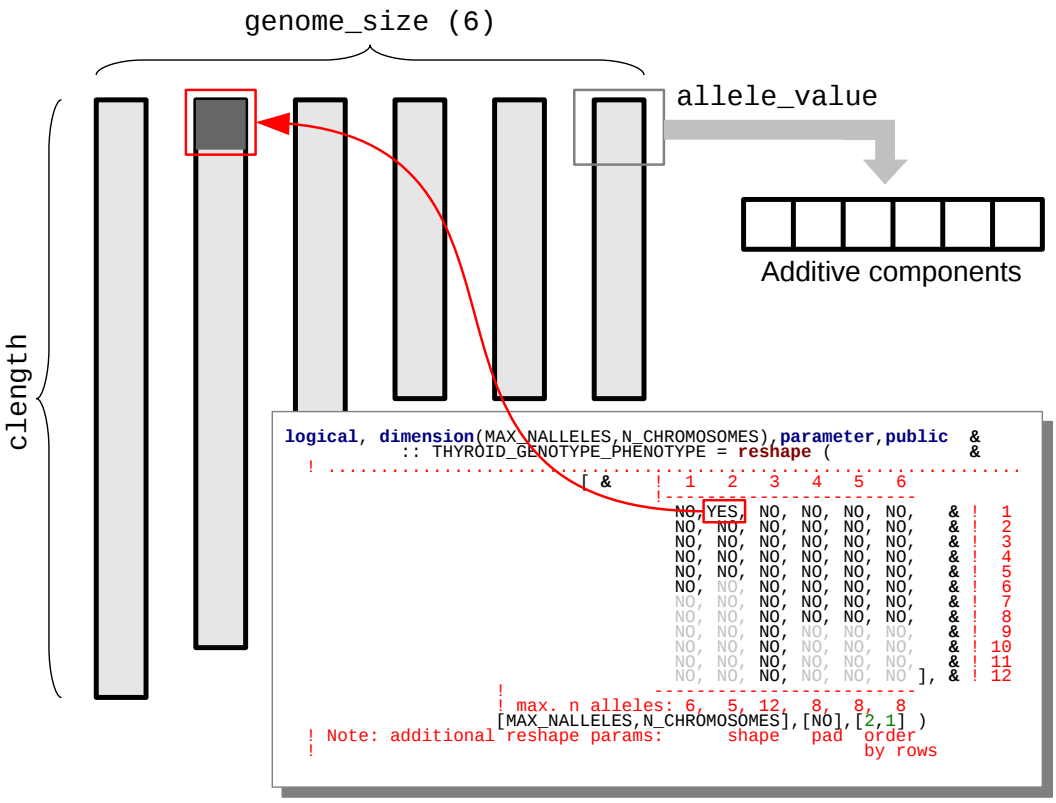


Figure 1.4 Genotype x phenotype matrix

An example of such a structure is the genetic sex determination: commondata::sex_genotype_phenotype. There is a small utility script `tools\gpmatrix.tcl` that assists in automatic production of the Fortran code for such a matrix.

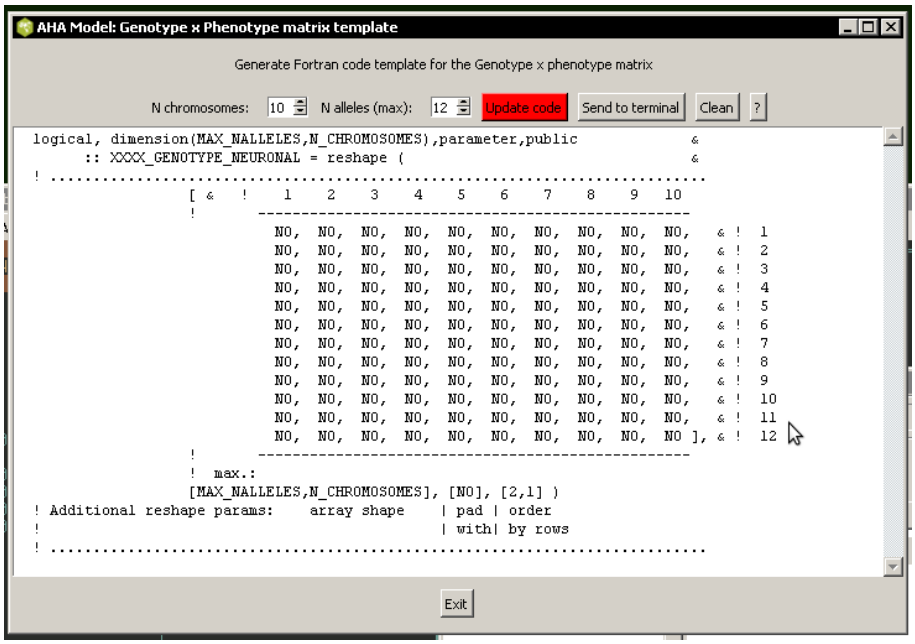


Figure 1.5 gpmatrix.tcl utility

Remarks

Windows distribution for Tcl/Tk language is obtained here).

1.4.2.3 Individual genome

The the_genome::individual_genome class defines the basic higher-level component properties of the whole organism, such as sex (logical type the_genome::individual_genome::sex_is_male), genome size (the_genome::individual_genome::genome_size), individual string "name" of the agent (the_genome::individual_genome::genome_lab
etc.
The the_genome::individual_genome class also includes such type-bound procedures as the_genome::individual_genome::lives() that gives the agent the "alive" status, the_genome::individual_genome::dies() for making the agent "dead". It also has linked procedures implementing genetic crossover

- the_genome::individual_genome::recombine_random(),
- the_genome::individual_genome::recombine_partial(),
- the_genome::individual_genome::crossover().

1.4.3 The individual agent

The individual agent has a the_environment::spatial_moving class as its base (but this class is an extension of the simple the_environment::spatial) and is then composed of several layers by class extensions.
Each of these main layers that create the individual agent is defined in separate Fortran module:

- genome (the_genome module);
- hormones (the_hormones module);
- the body characteristics and condition (the_body);
- neurobiological architecture (the_neurobio);
- behaviour architecture (the_behaviour) that builds on the neurobiology;
- finally, the agent is a member of a population (the_individual and the_population::member_population).

Class hierarchy: The *individual agent* and the *population* of agents

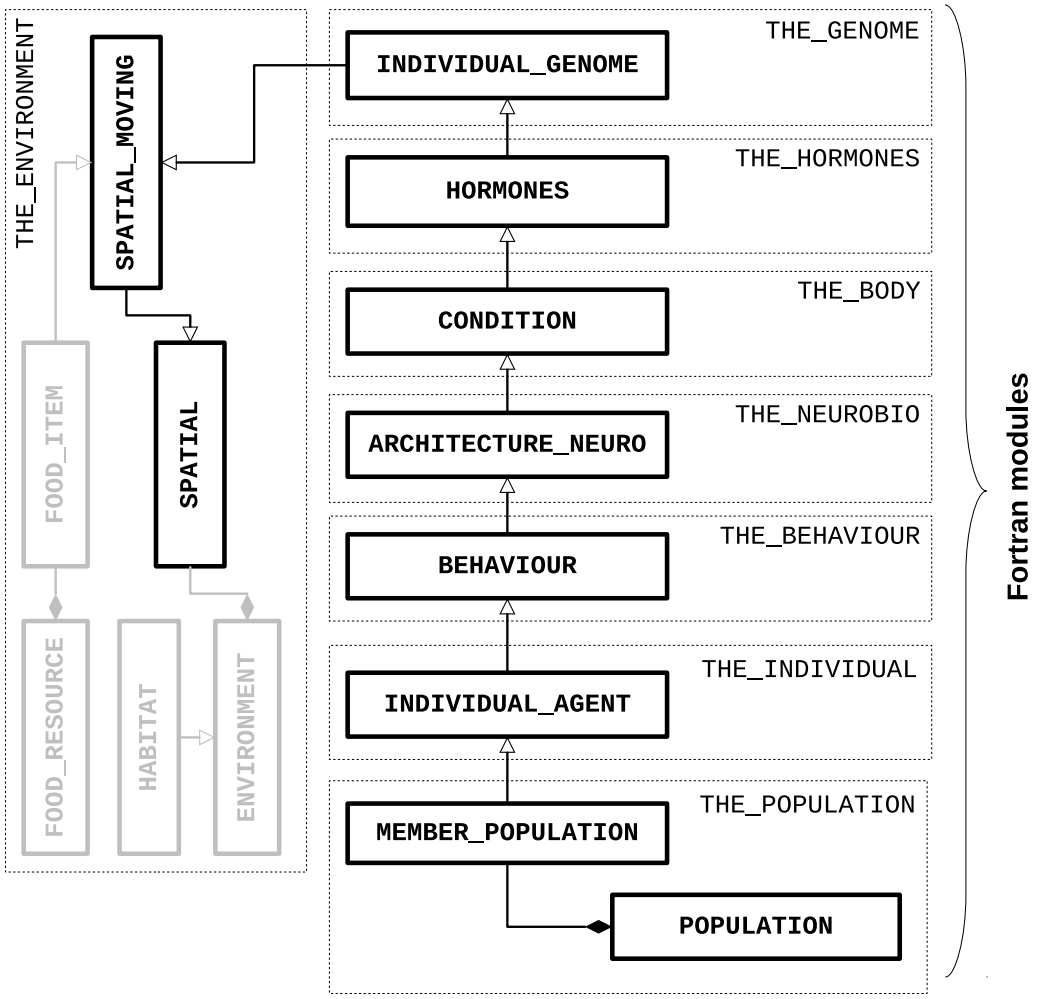


Figure 1.6 The individual class layers

The model code benefits from Fortran intrinsic "elemental" and array-based procedures. To initialise a whole population of random agents (with full object hierarchy) use the the_population::population::init() method:

```
call generation_one%init(POPSIZE)
```

Invocation of the 'init' method calls a whole cascade of various elementary object-bound procedures that create the whole population object.

1.4.4 Localist interactions

The model is based on (almost) fully proximate and localist philosophy. All objects (agents, predators, prey) can obtain information about only those objects that in proximity. No one is considered omniscient. Thus, objects can only interact locally, e.g. the agent can eat only food items that it could see, a predator could only attack agents that are visible to it etc.

The distance at which the objects can get information about each other is based on the visibility (visual range). Thus, the agents and predators are considered to be fully visual creatures that can only sense the world with their vision.

Visibility, i.e. the distance from which an object can be detected ("seen") depends on the ambient illumination at a specific depth, the area of the object, its contrast etc. Visual range is calculated for the different kinds of objects using the the_environment::visual_range() backend.

Examples of the visual range are the visibility distance of an agent the_body::condition::visibility(), food item the_environment::food_item::visibility(), and predator the_environment::predator::visibility().

Importantly, the perception of various kinds of environmental objects by the agent uses the the_environment::visual_range() calculation engine.

Furthermore, probabilities of stochastic events typically have non-linear relationships with the visual range. One

example is the probability of capture of a food item by the agent. This probability is high in close proximity but strongly reduces at the distance equal to the visual range limit: the_environment::food_item::capture_probability(). Similarly, the risk that the agent is killed by a predator is highest at a small distances and significantly reduces at a distance equal to the visibility limit (i.e. the maximum distance the predator can see the agent)↩ : the_neurobio::perception::risk_pred(). A more complex procedure is implemented for the probability of successful reproduction: the_neurobio::appraisal::probability_reproduction().

1.4.5 The Cognitive Architecture

The cognitive architecture of the agent is represented on the scheme below. It includes several functional units, "bundles," each representing specific motivation or emotion state.

Note

Note that the scheme below includes three such bundles for the states *A*, *B*, *C*. For simplicity, there are also only three stimuli S_1, S_2, S_3 .

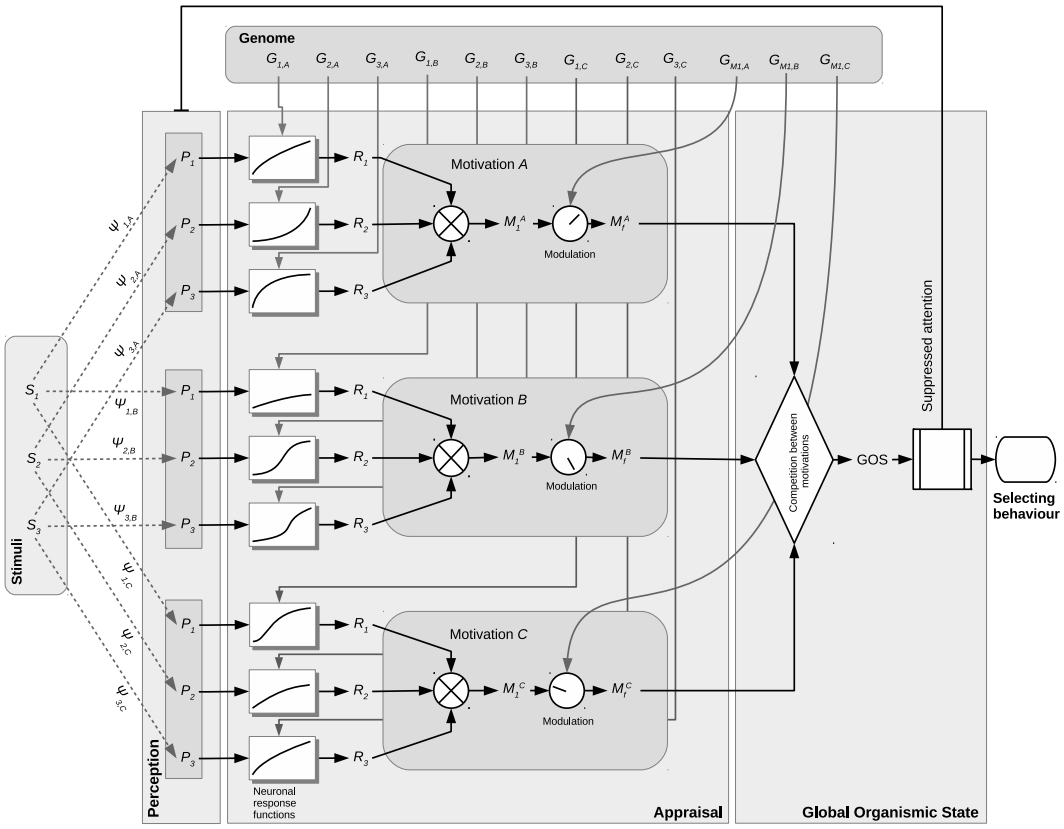


Figure 1.7 General cognitive architecture of the agent

General. The agent perceives the outer and its own inner environment, obtaining perception (signal) values. The agent also has several motivation (emotional) states, such that only one can be active at any time step of the model. Thus, the states compete for this position. The winning motivation (emotion) becomes the dominant emotional state of the agent, its Global Organismic State. This Global Organismic States of the agent determines how the agent weights different options during its decision making process and therefore determines what kind of behaviour (action) it will execute.

Perception and appraisal. The agent obtains perceptions (*P*) from its external and internal environments. These perceptions are fed into the Appraisal modules, separate for each of the motivation/emotion state.

Here perception signals are first fed into the neuronal response functions (based on the sigmoidal function commondata::gamma2gene()). The neuronal response function is a function of both the perception signal (*P*) and the genome (*G*) of the agent. Perception signal is also distorted by a random Gaussian error. Each neuronal response function returns the neuronal response (*R*) value.

The neuronal responses *R* for each stimulus are summed for the same motivation module to get the *primary motivation* values (M_1) for this motivational state. These primary motivations can then be subjected to genetic or

developmental (e.g. age-related) **modulation**, resulting in the *final motivation* values (M_f). Such modulation could strengthen or weaken the motivation values and therefore shift the outcome of competition between the different motivational states. In absence of modulation $M_1 = M_f$.

Global Organismic State. Final motivations (M_f) for different motivations (emotions) are competing, so that the winning state that is characterised by the highest final motivation value becomes the dominant emotional state of the agent: its Global Organismic State at the next time step. Additionally, the final motivation value of this state becomes the *GOS arousal* level.

The competition mechanism is complex and dynamic. It depends on the current arousal level, such that relatively minor fluctuations in the stimuli and their associated motivation values are ignored and do not result in switching of the GOS to a different state.

Furthermore, the relative difference (surplus) that the competing motivation must have to win competition against the current state depends on the current level of GOS arousal. If the current arousal level of the agent is relatively low (motivation or emotion is weak), a competing state must exceed a relatively higher threshold to win. However, if the current arousal level is very high (high motivation or emotion), a competing state can win even if it only slightly exceeds the current arousal level. Thus, the emotional state of the agent is characterised by a degree of continuity or "inertia" and such inetria is lower the higher is the current level of arousal. The dynamic threshold mechanism for GOS competition is described in details in the_neurobio::gos_find_global_state() procedure documentation section.

Attention focus. Whenever the agent has a specific Global Organismic State, this state also affects the agent's *perception*. All the perception inputs that belong to motivations other than the currently dominant (i.e. the current GOS) are suppressed by *attention weights*. For example, if the motivation *B* is the GOS, all the perception values linked with Motivation *A* and *C* are suppressed. The suppression weights are proportional to the current GOS arousal of the agent.

Thus, the attention mechanism effectively filters out or "focus" the agent on the stimuli that are linked with the current dominant emotional state. Moreover, the stronger is the current GOS arousal, the stronger is such attention focusing. Attention weight mechanism is described in details in the the_neurobio::gos_attention_modulate_weights() procedure section.

Perception-to-arousal path. This process, *perception* $\rightarrow$ *neuronal response* $\rightarrow$ *motivation* $\rightarrow$ *GOS* $\rightarrow$ *arousal* is repeated at each time step of the model as the agent acts in (e.g. moves through) its stochastic environment. In effect, the dominant motivational and emotional state of the agent changes adapting to to the latest changes in the inner and external environment.

Self-predictive decision making. Furthermore, the same processes (and computer code procedures) are also evoked when the agent is *making the decision* about what *behavioural action* to choose at each time step.

Basically, the agent predicts what would be its perceptions and, for each of the behavioural action available, runs the same process (perception $\rightarrow$ neuronal response $\rightarrow$ motivation $\rightarrow$ GOS $\rightarrow$ arousal) and finally selects the behaviour that would result in the lowest predicted GOS arousal. Perceptions in this process are predicted from the agent's internal or local external environment ("fake" perceptions in the `do_this` method for each of the behaviour units). They are also subjected to attention suppression, however, attention weights are transferred from the agent's own current Global Organismic State by the_behaviour::behaviour_base::attention_transfer() method.

Thus, decision making of the agent is based on predicting one's own emotional state. The emotional arousal therefore becomes a common currency in decision making. See Predictive decision making for more details.

Goal directed behaviour. The cognitive architecture implemented in the AHA model effectively produces goal-directed behaviour in the agents. The "goal" is defined by the evolutionary process, the Genetic Algorighm. Specifically , the target "goal" is to survive, grow, accumulate energy and reproduce. The agents that do best in this respect pass their genes into the next generation.

1.4.6 The Genetic Algorithm

1.4.6.1 Fixed explicit fitness GA

This version of the Genetic Algorithm (GA) is based on an explicitly defined "fitness" value for each individual agent. The evolution then optimises agents with respect to this fitness: only agents with high fitness values pass their genes into the next generations.

This algorithm for this kind of GA is simple and is implemented in the the_evolution::generations_loop_ga() procedure:

- Initialise the first generation of agents (population) from random genomes. This first generation becomes the "parents".
- Start the main GA generations loop
  - All "parent"-population agents go via the full life cycle (with many time steps), the_population::population::lifecycle_step().

  - Fixed fitness is calculated for each individual agent showing how good it did during the life cycle by the the_individual::individual_agent::fitness_calc() method.
  - All agents are sorted according to their fitness by the_population::population::sort_by_fitness().
  - Some number of the "best fitting" agents (i.e. those with the highest fitness values) are selected for the new generation: their genomes pass to the new "offspring" generation in an unchanged form (i.e. this is an *elitism*-based GA algorithm): the_evolution::selection().
  - The rest of the offspring population are obtained from the genomes of the best fitting agents, however,
    * parent's genomes randomly exchange their genetic material,
    * the resulting offspring genomes are subject to random mutations.

These steps are implemented in the_evolution::mate_reproduce().

- This new offspring population now becomes the "parents". A mechanism based on pointer swapping implemented in the_evolution::generations_swap() avoids copying big arrays of agents.
- Finally, the algorithm goes to the next generation loop cycle.

1.4.7 Life cycle of the agent

1.4.7.1 Initialisation

The life cycle of the agent begins with birth. In the first generation of the Genetic Algorithm each agent is initialised from a random genome by calling the the_individual::individual_agent::init() method. This method evokes a cascade of procedures that initialise all levels of the individual agent object hierarchy.
All subsequent generations use pre-existing genomes that are passed from the ancestral agents subject to genetic crossover and mutation. The the_individual::individual_agent::init() procedure has a logical switch that controls if a random genome should be generated.
Because the population of agents is an object (class the_population::population), a whole population of agents is initialised by the the_population::population::init() method. After the birth, all agents get random location within their initial the_environment::habitat object by calling the_population::population::scatter_uniform().

1.4.7.2 Life cycle step

Then, the agents pass through many time steps (commondata::lifespan). At each time step each agent gets internal and environmental perceptions. Based on these instantaneous perceptions, they obtain their main internal state: the Global Organismic State (GOS). Finally, each agent must make a decision to execute a single specific behaviour depending on its GOS, internal and external environment (perception). Some examples of such behaviour are eat a food item, approach a conspecific, escape from a predator, move, freeze, migrate to a novel environment, reproduce, etc.
Each of the behaviour chosen by the agent has specific consequences and can

- affect the agent (e.g. incurs energetic cost, or relocation of the whole agent into a novel environment, the agent is killed if it does not escape a nearby predator),
- affect the other agents in proximity (e.g. emigration of an agent could change the risk of predation for other agents by changing the spatial configuration and therefore modifying predation dilution and confusion effects for the agents that remain in place),
- affect the external environment (e.g. food item disappears when an agent eat it).

Whenever the agent successfully consumes food items, it grows its mass and length: the_body::condition::mass_grow(), the_body::condition::len_grow(). On the other hand, every movement incurs some energetic cost: the_body::condition::cost_swim(). There are also additional costs linked with specific behavioural actions, e.g. food processing cost the_body::condition::food_process_cost() or the cost of successful (the_body::reproduction::reproduction_cost()) or unsuccessful (the_body::reproduction::reproduction_cost_unsuccess()) reproduction.
There is also an overall living cost that is subtracted (the_body::condition::subtract_living_cost()). Additionally, digestion occurs by emptying the agent's stomach by a fixed fraction (the_body::condition::stomach_empify() at each time step.

1.4.7.3 Predation and mortality

At each time step, the agent can be subjected by random disastrous events, such as habitat-specific random mortality the_population::population::mortality_habitat() or predatory attacks the_population::population::attacked(). If the predatory attack is successful, the agent the_genome::individual_genome::dies() and cannot normally pass its genome into the next generation.
The agents are also checked at various stages of their life cycle for starvation: the_body::condition::starved_death(). If the agent is starved, it also the_genome::individual_genome::dies().
Predation risk (the probability of the agent being caught by the predator) is automatically adjusted if the predator can simultaneously perceive several conspecific agents. Depending on the number of such agents in the perceived group and individual distances between the predator and each agent, a predator dilution and confusion effects are calculated (see the_environment::predator::risk_fish_group()). By the way, when the agent makes the decision to approach conspecific in presence of a predator, it also evaluates how many other conspecifics are there and what is the relative distances between the conspecifics and the predator (see the_behaviour::approach_conspec::do_this()). The agent's decision therefore adjusts for the classical "selfish herd" effect.

1.4.7.4 Food competition

Because many agents are dwelling within the same habitat with limited food resource (fixed number of the_environment::food_item objects), they also compete for food. For example, if two agents find themselves near the same food item, one or the other can eat it, whoever is the first to make the decision to the_behaviour::eat_food. If an agent successfully eats such food item, it is marked as the_environment::food_item::eaten and is not available for everyone else.
Because the model is localist and explicitly implements individual capture of each food item by each agent within spatially explicit habitats, any density- and frequency-dependent effects would appear automatically.

1.4.7.5 Reproduction

Whenever the agent the_body::reproduction::is_ready_reproduce(), it can make decision to the_behaviour::reproduce at a specific time step. The probability of successful reproduction in specific stochastic conditions is calculated by the_neurobio::appraisal::probability_reproduction(). As a consequence of successful reproduction, some number (the_body::reproduction::offspring_number()) of offspring are produced. How subjective value of reproduction is evaluated by the agent is dictated by the_behaviour::reproduce::do_this().

1.4.7.6 Agent order and competition

To avoid systematic biases, the agents make their behavioural choices in a random order (see `PERMUTE_RANDOM()`). However, it is also easy to implement any other sorted order, e.g. selecting agents according to their body weight: the body weight then would provide a strong competitive advantage.
Any arbitrary order of the agents can be implemented using `array indexing` mechanism. See the_population::population::lifecycle_step() for more discussion.

1.4.7.7 Concluding remarks

The long sequence of such decision makings in a stochastic environment at each time step constitutes the life cycle of the agent. Depending on how well the agent does during its life cycle (e.g. grows and reproduces) determines the chance that its genome passes to the next generation by the Genetic algorithm. Thus, the whole process simulates the behaviour, decision making and evolution.
The following sections provide general information about the implementation and the object class hierarchy of the model code.

1.4.8 The perception mechanism

1.4.8.1 Overview

Perception is defined in the the_neurobio::perception class. Perception objects can be of three types:

- Internal perception objects, depend on the body.
  - the_neurobio::percept_stomach

  - the_neurobio::percept_energy
  - the_neurobio::percept_age
  - the_neurobio::percept_reprfact

- Direct environmental perceptions (e.g. light).
  - the_neurobio::percept_light
  - the_neurobio::percept_depth

- External spatial perception objects, depend on the visual range.
  - the_neurobio::percept_food
  - the_neurobio::percept_predator
  - the_neurobio::percept_conspecifics

Class hierarchy: *perception objects*

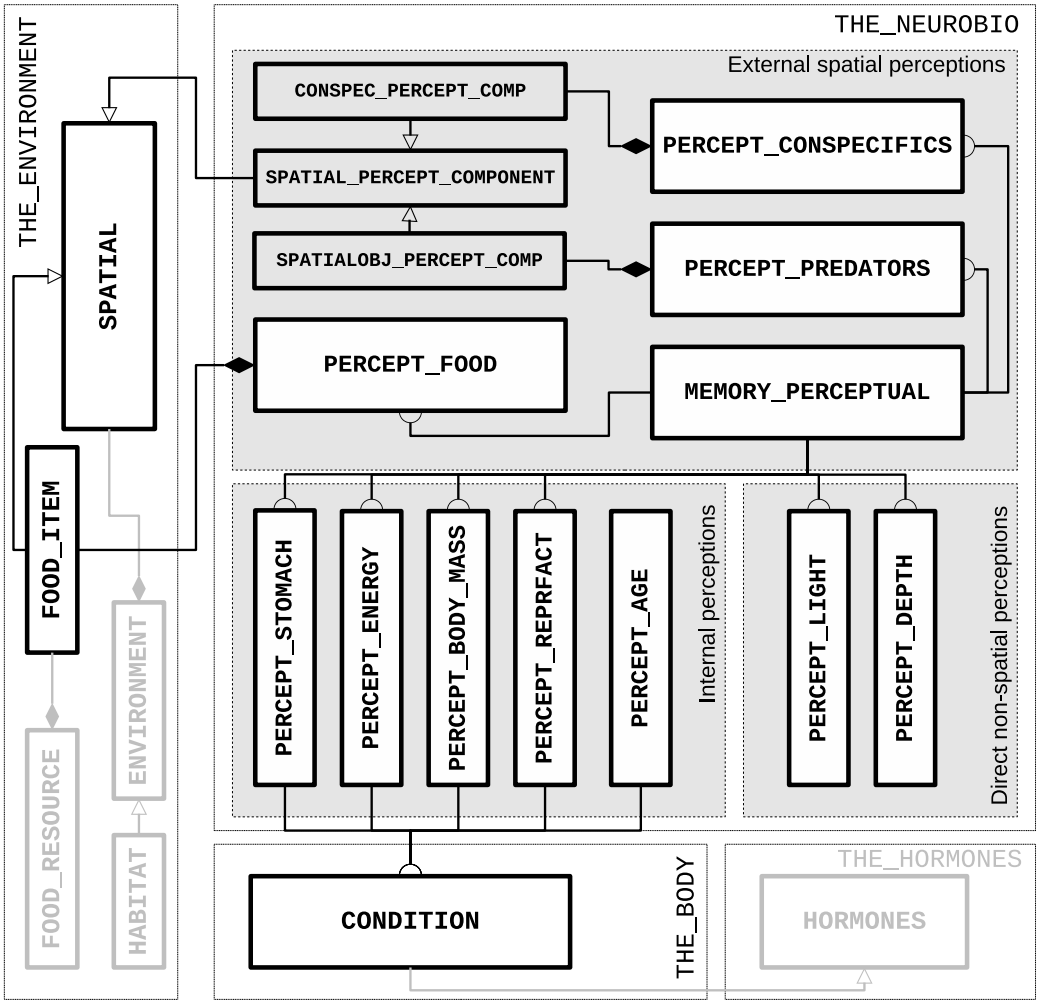
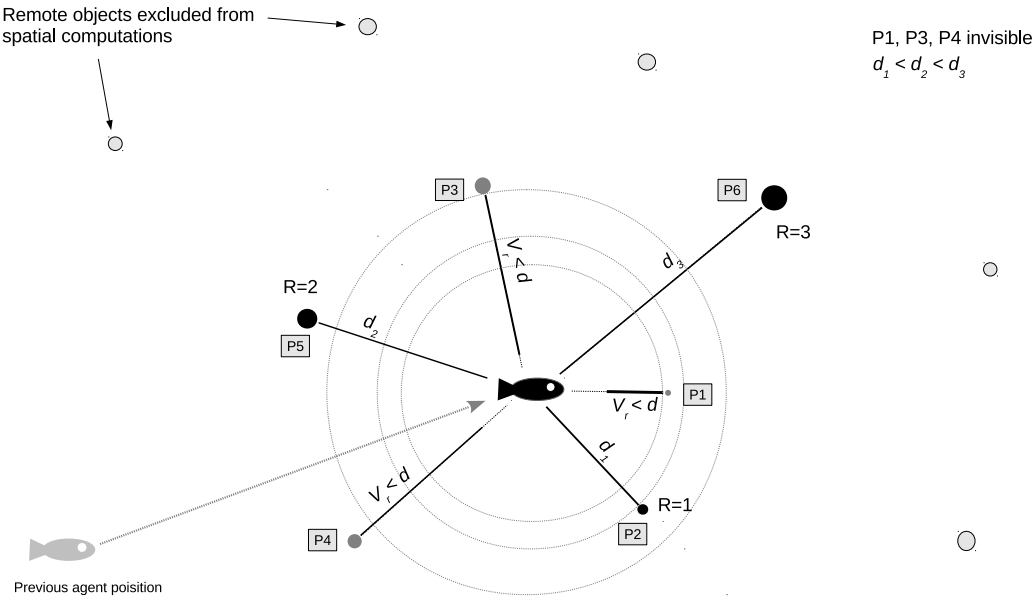


Figure 1.8 Perception objects

1.4.8.2 Spatial perceptions

External spatial perception components are truly "localist" and proximate, they get individual environmental objects (e.g. food items, individual conspecifics or predators) dynamically as the agent and the spatial objects move within the model 3D environment. This allows aha_buildblocks_gp_matrix_introto produce very complex environmental structures (e.g. patchy foods with Gaussian scatter). Also, every individual environmental object is perceived only

if it is within the specific visual range. This means that, as food items are stochastic (have random Gaussian size), perception of each food item depends on its individually-specific visibility range. The same is true also for conspecifics, predators, etc.



- **Spatially explicit:** as the agent is moving in a 3D space at each time step, the perception system accesses various stochastic spatial objects around it, e.g. food items, predators, conspecifics. For each of these objects, the visibility (visual range) is calculated and the agent "sees" only thode objects that are within it.
- **Spatial segmentation:** Only a limited number of spatial neighbouring objects are assessed, the other are ignored by a partial distance ranking algorithm ensuring fast calculatuion

Figure 1.9 Spatial perception

Selection of the nearest environmental objects that are within the current visual range (the_environment::spatial::neighbours()) is based on partial indexing (spatial segmentation) of potentially huge arrays of different objects (e.g. thousands of individual stochastic food items, each with specific visual range). Partial indexing allows very fast processing of only a subset of spatial objects that are in proximity of the agent (and therefore could fit into the visibility range), ignoring al other, more remote, objects in the same environment.

1.4.9 From perception to GOS

The overview below shows the sequence of the main procedures from perception (the_neurobio::perception class) through the_neurobio::appraisal to the determination of the Global Organismic State (GOS, the_neurobio::gos_global class).

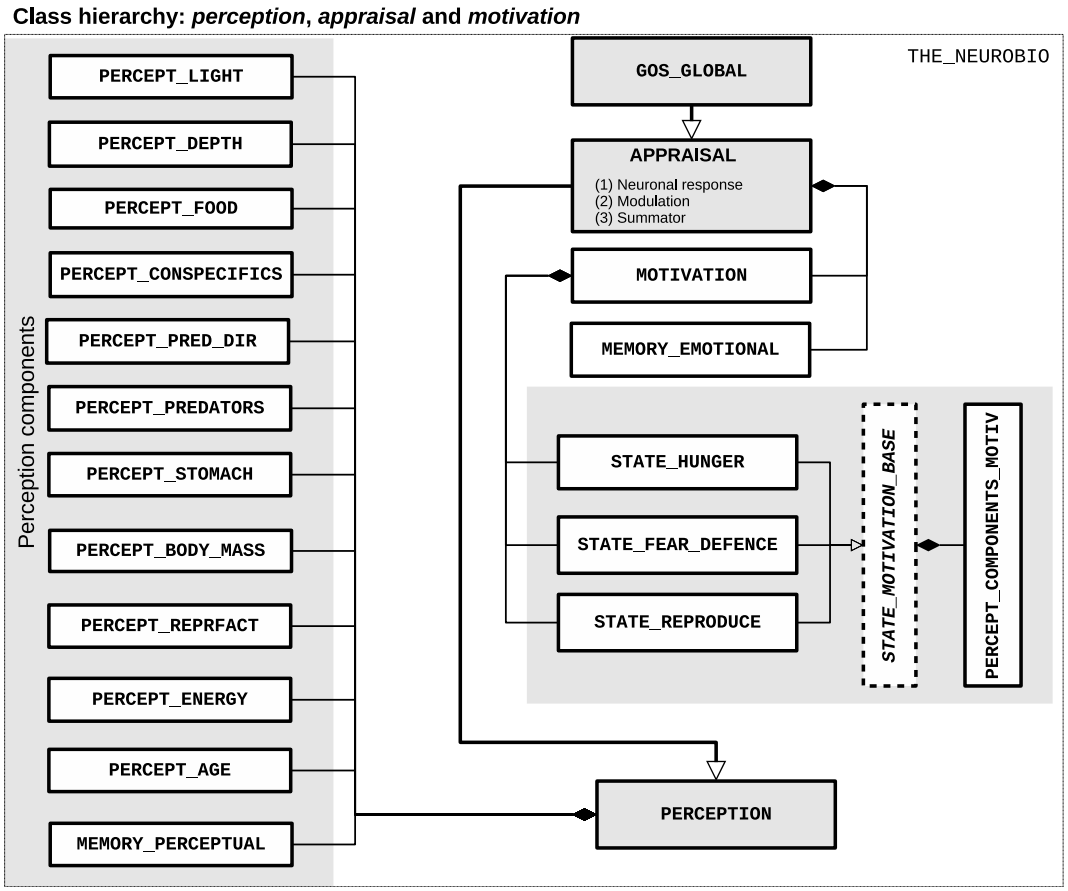


Figure 1.10 Appraisal objects

1.4.9.1 Perception

Perception: first, the agent obtains perceptions from its inner environment (such as age, stomach contents etc), the external environment (e.g. light, depth), as well as spatial perceptions (e.g. food items, conspecifics, predators). These *perception components* are described by the the_neurobio::percept_components_motiv class. Perception components represent a crucial component of each of the *motivational states*. Perceptions are obtained by calling these procedures:

- the_neurobio::perception::perceptions_inner() - inner perceptions (an umbrella for several perceptions), these include:
  - the_neurobio::perception::feel_stomach()
  - the_neurobio::perception::feel_bodymass()
  - the_neurobio::perception::feel_energy()
  - the_neurobio::perception::feel_age()
  - the_neurobio::perception::feel_repfac()
- the_neurobio::perception::perceptions_environ() - environmental perceptions (an umbrella procedure for several perceptions) including
  - the_neurobio::perception::feel_light()
  - the_neurobio::perception::feel_depth()
- the_neurobio::perception::see_food() - get food;
- the_neurobio::perception::see_pred() - get predators;
- the_neurobio::perception::see_consp() - get conspecifics;

Notably, the spatial perceptions dynamically access stochastic external spatial objects using a fast algorithm based on partial distance ranking of spatial objects, a kind of spatial segmentation. This largely reduces the need to multiply loop across the objects and the agents over many time steps and generations.

1.4.9.2 Appraisal

Appraisal: perceptions are weighted by the *attention weights* and go into the *neuronal response* function the_genome::individual_genome::neuro_resp(), then summed to get the *primary* motivation value for each of the motivation states.

1.4.9.2.1 Neuronal response function The neuronal response functionhas a central role in the model. It links the sensory perception from a specific environmental stimulus *P* (including the inner organism's environment), and the strength of the response to this stimulus (the neuronal response) *R* mediated by the genome. Neuronal response function is based on the sigmoidal equation that is implemented in commondata::gamma2gene() procedure.
Note that this function is not based on a mechanistic understanding of the relationship between perception and neuronal response. Lacking such understanding, the genetic algorithm may produce a series of potentially adaptive relationships. Depending upon the allele values of the genes, the function can appear concave, sigmoidal, nearly linear, or convex in the *0 P 1* range. We use a sigmoidal function to avoid total lack of (emotional) interest at very low *P*.

1.4.9.2.2 From neuronal responses to motivations and emotions There are currently four motivational states that the agent can have:

- the_neurobio::state_hunger
- the_neurobio::state_fear_defence
- the_neurobio::state_reproduce

All these motivational states represent extensions of the basic *abstract class* the_neurobio::state_motivation_base that is not used directly.
Calculating the *primary* motivational values involves calling these procedures:

- the_neurobio::appraisal::motivations_percept_components() – perceptions are fed into the neuronal response function, obtain perceptual components for each of the above motivational states ;
- the_neurobio::appraisal::motivations_primary_calc() – perception components are fed into this procedure resulting in calculation of *primary* motivations, i.e. values of the above motivational states before they are subjected to any modulation.

These primary motivation values for each of the above four motivation states are subjected to genetic or developmental modulation, resulting in *final* motivation value for each of the motivational states:

- the_neurobio::appraisal::modulation();

They are also recorded into the emotional memory stack by calling

- the_neurobio::appraisal::motivations_to_memory();

1.4.9.3 GOS

GOS: finally, a single *Global Organismic State* (GOS) is determined on the basis of the competition between the final values of all motivational states. GOS is determined in this procedure:

- the_neurobio::gos_global::gos_find();
  - gos_find() also internally calls the attention modulation procedure the_neurobio::gos_global::attention_modulate() that limits attention all perceptions that are "irrelevant" for the current GOS;

1.4.9.4 Code example

The code below is an example of the above steps, from perception to appraisal and GOS:

```
! Perception:
call proto_parents%individual(ind)%perceptions_inner()
call proto_parents%individual(ind)%perceptions_environ()
call proto_parents%individual(ind)%see_food( food_resource )
call proto_parents%individual(ind)%see_consp( proto_parents%individual )
call proto_parents%individual(ind)%see_pred( habitat_safe%predators )
! Add perceptions to the memory:
call proto_parents%individual(ind)%perception_to_memory()
! Appraisal:
call proto_parents%individual(ind)%motivations_percept_components()
call proto_parents%individual(ind)%motivations_primary_calc()
call proto_parents%individual(ind)%modulation()
call proto_parents%individual(ind)%motivations_to_memory()
! GOS:
call proto_parents%individual(ind)%gos_find()
```

1.4.10 The behavioural repertoire

The *behavioural repertoire* of the agent is composed of several *behaviour units*, which are extensions of the basic the_behaviour::behaviour_base class (this is an abstract class and is not used directly). Thus, each component of the behavioural repertoire is a *separate object* `class` not linked with the *agent* class hierarchy (from the_genome::individual_genome up to the_neurobio::gos_global).
However, all the individual components of the behavioural repertoire are collected together in the the_behaviour::behaviour class (that is in the agent class hierarchy). Thus, the behavioural repertoire of the agent is here *constructed* from individual behavioural components.

Class hierarchy: behavioural repertoire

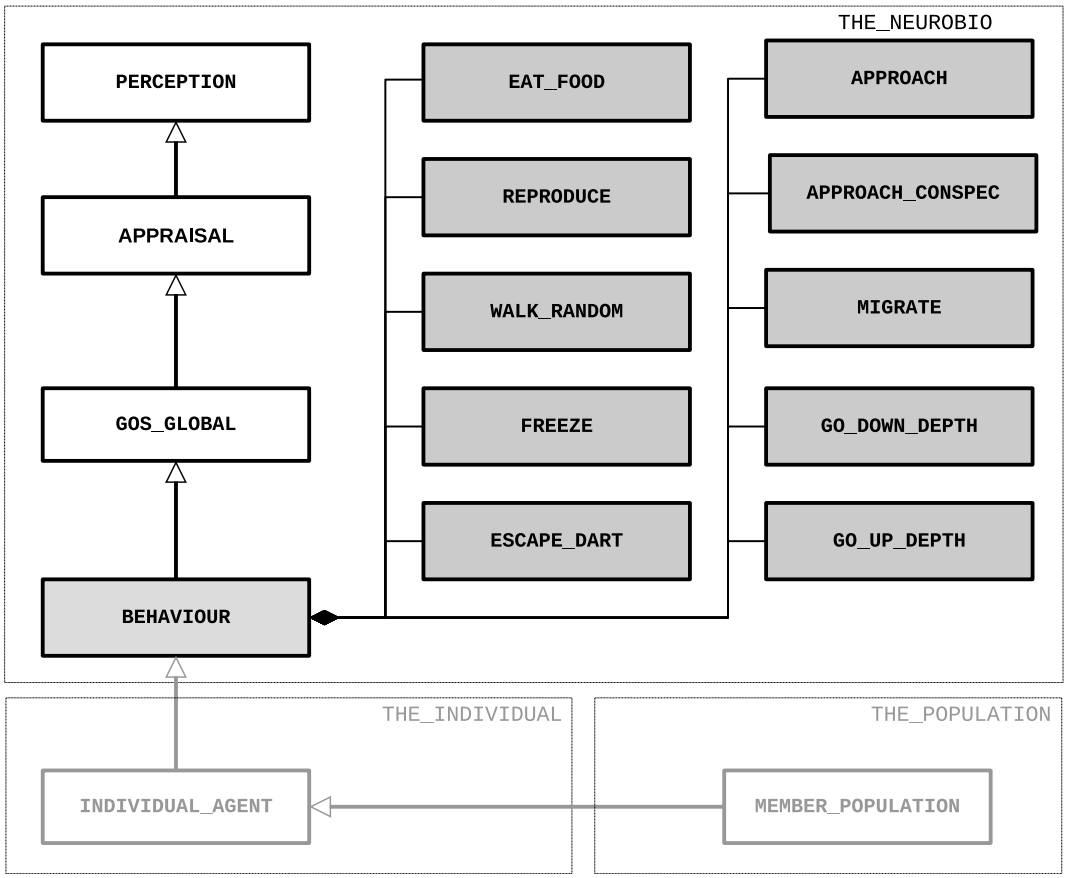


Figure 1.11 Construction of the behaviour

1.4.10.1 Behavioural units

Here is the behavioural repertoire of the agent:

- the_behaviour::eat_food

- the_behaviour::reproduce
- the_behaviour::walk_random
- the_behaviour::freeze
- the_behaviour::escape_dart
- the_behaviour::approach
- the_behaviour::approach_conspec
- the_behaviour::migrate
- the_behaviour::go_down_depth
- the_behaviour::go_up_depth

The inheritance structure of these behavioural units is shown on the scheme below.

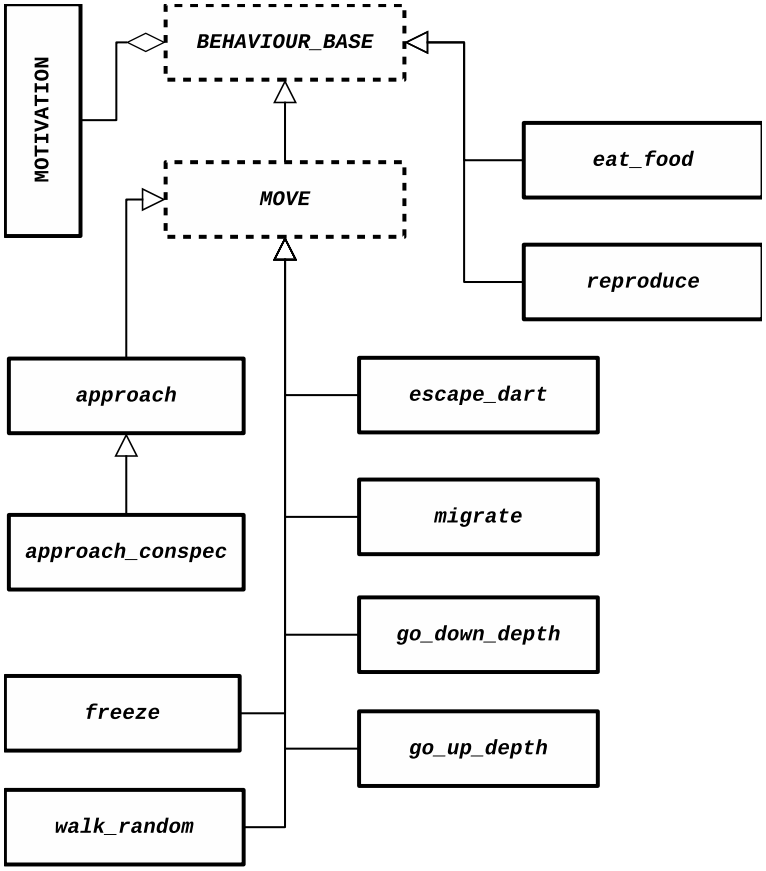


Figure 1.12 The behaviour repertoire

Some of the behaviours involve "movement", i.e. change of the spatial location. Such behaviours are programmed as extensions of the base class the_behaviour::move (this is also an abstract class that is not used directly).

- the_behaviour::walk_random
- the_behaviour::freeze
- the_behaviour::escape_dart
- the_behaviour::approach
- the_behaviour::approach_conspec (an extension of the_behaviour::approach)
- the_behaviour::migrate

- the_behaviour::go_down_depth
- the_behaviour::go_up_depth

Each of these *movements* can differ by its *distance* or length (although the_behaviour::freeze always has zero distance): the_behaviour::move::distance. Thus, there can be a whole repertoire of each of the above movements, such as Gaussian random walks or vertical migrations with different steps.
the_behaviour::eat_food also involves movement but indirectly, so it is not related to the the_behaviour::move base class.
Finally, certain behaviours involve specific "target object":

- the_behaviour::eat_food
- the_behaviour::escape_dart
- the_behaviour::approach
- the_behaviour::approach_conspec
- the_behaviour::migrate

Such a target can be, for example, food items (the_environment::food_item), predators (the_environment::predator), conspecifics (the_neurobio::appraisal) or an environmental container (the_environment::environment).
Combinations of several behavioural units with multiple *distances* (for movements) and *targets* creates an additional diversity, complexity and flexibility of the behaviour in the agents.

1.4.10.2 Double role of motivation

Notably, the the_neurobio::motivation class is included both into the **agent's** the_neurobio::appraisal (motivations) as well as into the **behaviour unit** the_behaviour::behaviour_base (expectancy).
Such a complex design composed of separate behavioural units (separate classes although inheritances of the same base the_neurobio::motivation) makes it possible to generate behaviours that:

- depend on inputs from the_neurobio::perception of the agent, both "objective" perceptions that flow from the environment and "subjective" perceptions that are internally generated as a part of the subjective evaluation of various possible outcomes of behaviour;
- link **perception** to **motivation** both in the agent itself and in each of the behavioural units;
- have multiple behaviours with their linked motivational expectancies that are collected (plugged) back into the agent class hierarchy;
- so that the agent can compare the motivational expectancies from each of the possible behaviour units, also in response to each of the multiple perception targets (e.g. specific food items, conspecifics and predators);
- the agent can select the behaviour (and the specific target) that depends on the expected arousal, for *execution*;
- finally, this optimal behaviour is "executed" (with its specific target) by calling its specific `do_` method in the the_behaviour::behaviour class.
- exactly the same class data structures and procedure(s) are employed to calculate the motivation values:
  - for the the_neurobio::appraisal::motivations of the **agent** and
  - for motivational the_behaviour::behaviour_base::expectancy of the each **behaviour unit**.

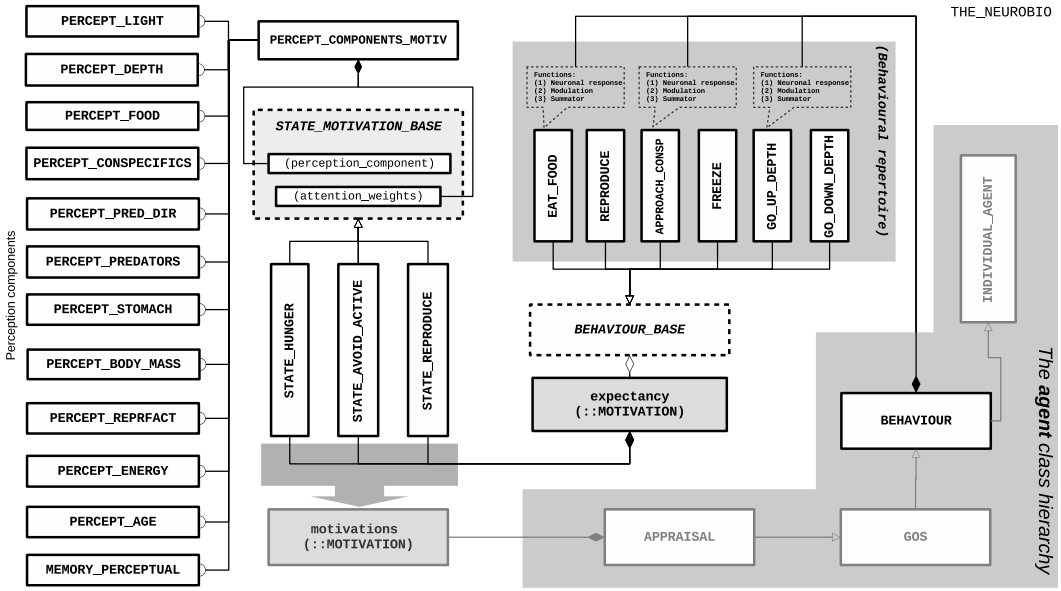


Figure 1.13 Motivation and behaviour

1.4.11 Predictive decision making

1.4.11.1 Overview

Decision making in the AHA Model is centered around three concepts:

- motivation (emotion)
- global organismic state (GOS)
- expected arousal

The main principle of decision maming is simple. It involves evaluating each of the available behavioural alternative (each behaviour unit with specific parameters and targets) to find those alternative which minimises the expected arousal. This option is then considered the *optimal* behaviour and and is finally *executed*. This is a kind of a "brute force" optimisation mechanism that aims to get the most optimal behavioiral outcome from the agent's knowledge of the local environment and its own internal state.
Thus, each agent evaluates the environment in the context and in terms of *its own predicted internal state*. Thie outcome of such a process therefore depends both on the local instantaneous environmental context (depicted by the agent's local perception of food, conspecifics and predators), and the current internal state of the agent (its current motivation, the global organismic state, GOS).
The emotional state of the agent represents the common currency, the GOS arousal, for predicting the optimal behaviour: selection of a single action out of numerous available alternatives.

1.4.11.2 The umbrella class

Selection and *execution* of the optimal behaviours is implemented in the the_behaviour::behaviour class, which is a pert of the individual agent class hierarchy. The the_behaviour::behaviour class collects all the separate behavioural units ans thereby serves an umbrella construict for the whole behavioural repertoire of the agent. It also includes high level execution procedures for each of the behavioural unit:

- the_behaviour::behaviour::do_eat_food_item();
- the_behaviour::behaviour::do_reproduce();
- the_behaviour::behaviour::do_walk();
- the_behaviour::behaviour::do_freeze();
- the_behaviour::behaviour::do_escape();

- the_behaviour::behaviour::do_approach();
- the_behaviour::behaviour::do_migrate();
- the_behaviour::behaviour::do_go_down();
- the_behaviour::behaviour::do_go_up();

These execution "do_" procedures basically call the "execute" methods for their respective specific behaviour unit (each behaviour unit is implemented as a separate class class, see Behavioural units). For example, the_behaviour::behaviour::do_eat_food_item() calls the_behaviour::eat_food::execute(). This provides a connection between the the_behaviour::behaviour umbrella class and each of the behavioural units. Furthermore, this connection allows to call a whole cascade of methods to assess and predict possible consequences of executing each of the possible behaviour.

1.4.11.3 do_behave method

The the_behaviour::behaviour class also implements the the_behaviour::behaviour::do_behave() method which provides a unitary interface for the evaluation, selection and execution of the optimal behaviour unit, i.e. the behaviour minimising the expected GOS arousal.
The the_behaviour::behaviour::do_behave() method

- Evaluates each of the available behavioural alternative given the specific current context (internal and external perception objects). This is implemented by the various lower order `select` procedures within `do_behave()`:
  - eat_food_select();
  - reproduce_select();
  - walk_random_select();
  - freeze_select();
  - escape_dart_select();
  - approach_consp_select();
  - migrate_select();
  - go_down_select();
  - go_up_select().

These `select` procedures calculate the arousal expectancy for each of the behavioural unit. If specific behaviour unit has multiple targets (e.g. several food items or several conspecifics) or parameters (e.g. random walks of different Gaussian length or vertical migrations with different step sizes), each such option is also evaliuated and the optimal one is determined (e.g. the "best" food item, eating which would result in lowest arousal).

- Once the expected arousal level is known for each of the behaviours, the procedure determines the one that minimises the GOS arousal overall.
- Finally, the behaviour (with specific optimal target or parameters) that would minimise the resulting GOS arousal is executed by calling respective `do_` procedure (e.g. the_behaviour::behaviour::do_eat_food_item() for the optimal food item if feeding was selected as minimising the overall arousal).

This process is schematically depicted in the figure below.

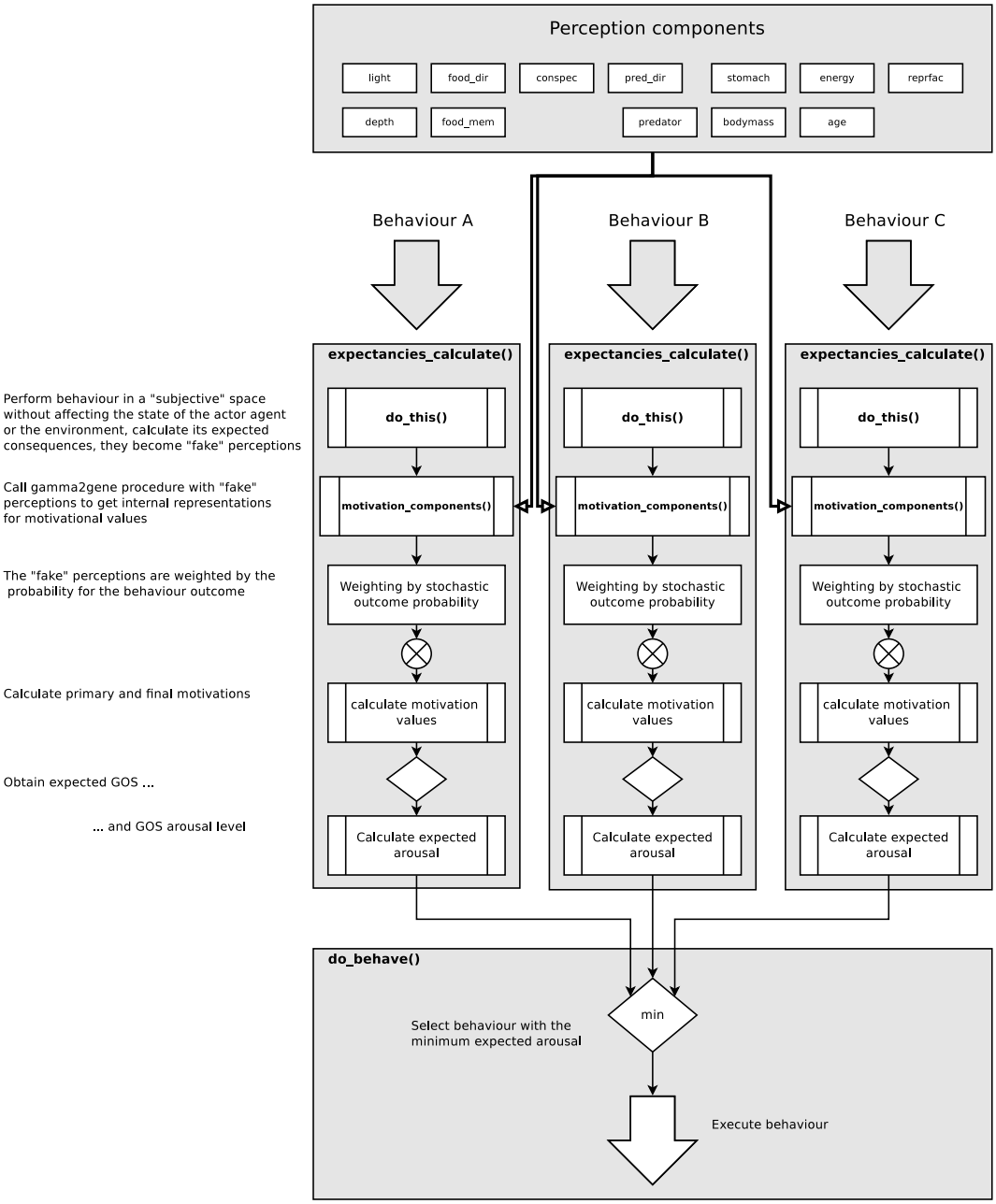


Figure 1.14 Predictive decision making by minimising expected GOS arousal

Chapter 2

Introduction and Getting Started

2.1 The AHA Model

This is a large scale simulation model (under development) that implements a general **decision-making architecture** in **evolutionary agents**. Each agent is programmed as a whole virtual organism including the genome, rudimentary physiology, the hormonal system, a cognitive architecture and behavioural repertoire. They "live" in a stochastic spatially explicit virtual environment with physical gradients, predators and prey. The primary aim of the whole modelling machinery is to understand the evolution of decision making mechanisms, personality, emotion and behavioural plasticity within a realistic ecological framework. An object-oriented approach coupled with a highly modular design not only allows to cope with increasing layers of complexity inherent in such a model system but also provides a framework for the system generalizability to a wide variety of systems. We also use a "physical-machine-like" implementation philosophy and a coding standard integrating the source code with parallel detailed documentation that increases understandability, replicability and reusability of this model system.
The cognitive architecture of the organism is based on a set of motivational (emotional) systems that serves as a common currency for decision making. Then, the decision making is based on **predictive assessment** of external and internal stimuli as well as the agent's own motivational (emotional) state. The agent makes a subjective assessment and selects, from the available repertoire, the behaviour that would reduce the expected motivational (emotional) arousal. Thus, decision making is based on predicting one's own internal state. As such, the decision-making architecture integrates motivation, emotion, and a very simplistic model of consciousness.
The **purpose** of the AHA model is to investigate a general framework for modelling proximate decision-making and behavior. From this we will investigate adaptive goal-directed behaviour that is both guided by the external environment and still is endogeneously generated.
Other research topics include individual differences, personality as well as consequences of emotion and personality to population ecology.
We think that understanding and modelling complex adaptive behaviour requires both extraneous (environmental) factors and stimuli as well as endogeneous mechanisms that produce the behaviour. Explicit proximate representation of the motivation and emotion systems, self-prediction can be an important component in linking environment, genes, physiology, behavior, personality and consciousness.
Fortran is used due to its simplicity and efficiency. For example, check out this paper.

2.2 Subdirectories of the AHA Model code

- `\dox` – contains pictures, plots and other resources for the full Doxygen documentation that is extracted from the source code.
- `\pfunit` – unit tests using pFUnit, so far rather rudimentary.
- `\tools` – various accessory tools, post-processing of the data generated by the model etc.

2.3 Getting started

Building and running the mode is based on the GNU Make. Both the **AHA Model** code and the **HEDTOOLS** (code or static library) are needed to build the model. These two components of the AHA Model framework are described in the Overview of the AHA Fortran modules.

Normally, the *AHA Model* code and the *HEDTOOLS* code are placed in two subdirectories of the working directory using two separate commands to get the code of the AHA Model and HEDTOOLS from the repositories (`svn co https://...` or `hg clone ssh://...` if Mercurial is used – check out Subversion).
With the current main Subversion repository, getting the code requires these commands:

```
svn co https://tegsvn.uib.no/svn/tegsvn/branches/budaev/HEDG2_01
svn co https://tegsvn.uib.no/svn/tegsvn/branches/budaev/HEDTOOLS
```

or these, if Mercurial is used (here local folders are capital):

```
hg clone ssh://hg@bitbucket.org/ahaproject/hedg2_01 HEDG2_01
hg clone ssh://hg@bitbucket.org/ahaproject/hedtools HEDTOOLS
```

Thus, the layout of the working directory after `HEDG2_01` and `HEDTOOLS` are downloaded is like this:

```
workdir
 |
 |-- HEDG2_01
 |     |-- dox
 |     |-- pfunit
 |     `-- tools
 |
 `-- HEDTOOLS
```

Building the AHA Model is done in the model directory (here `HEDG2_01`). If you use the terminal, go there with

```
cd HEDG2_01
```

Here are the main commands to build and run the AHA Model:

- *Build* the model from the source code: `make`;
- Build and *run* the model:`make run`;
- Delete all build-related, data and temporary files `make distclean`;
- Get a quick help from the make system: `make help`.

See `Makefile` for build configuration. To get more information on the GNU Make see `AHA Modelling Tools Manual`, `Using Microsoft Visual Studio` and `Using Code::Blocks IDE`.

2.4 Environment variables

The model makes use of several environment variables to control certain global aspects of the execution. These variables have the `AHA_` prefix and are listed below.

- `AHA_DEBUG=TRUE` sets the "debug mode";
- `AHA_SCREEN=NO` sets logger to write to the standard output as well as in the log file;
- `AHA_DEBUG_PLOTS=YES` enables generation of debug plots;
- `AHA_ZIP_FILES=YES` enables background compression of big output data files;
- `AHA_CHECK_EXTERNALS=NO` disables checking for external executable modules (this is a workaround against Intel Fortran compiler bug).

2.5 Makefile: GNU Make build configuration

`Makefile` to build the model executable from the source code.
This is a standard GNU Make `Makefile` to build the model from the sources. It also automatically generates some platform and compiler specific include files for the `BASE_RANDOM` and the IEEE math modules. See `Working with the model` section for details.
In addition to the GNU Make, this `Makefile` also depends on the `grep`, `cut`, `sed` and `awk` utilities that are installed on any Unix/Linux system, but usually absent on Windows. For Windows they can be obtained from several locations on the web, e.g. Cygwin or GnuWin32. See `AHA Modelling Tools Manual` for more information.

2.5.1 Main adjustable parameters

- **FC** sets the default compiler type. Normally GNU `gfortran` or Intel `ifort`.
- **HOST_HPC_ROOT** is the hostname to run the model executable in the HPC batch mode. If the hostname the `Makefile` is called in is this system, `make run` starts a new batch task. Otherwise, the model executable is just started normally.
- **SRC** is the name of the main source code (can be several files).
- **DRV** is the source code of the model "driver", that is the main Fortran program that produces the executable. It should be very very small. The "driver" is a separate file from the AHA Model modules.
- **OUT** is the executable file name that is actually run, on the Windows platform must have the `.exe` suffix. By default, the executable name is set to `MODEL.exe` for maximum portability.
- **DOXYCFG** the Doxygen documentation system configuration file name.
- **DOXYPATH** defines the Doxygen output directory where the documentation is generated. This could be set manually or parsed automatically using the `grep` command. Note that the output directory is set in the `Doxyfile` as: `OUTPUT_DIRECTORY = ./model_docs`.
- **HEDTOOLS** is the list of HEDTOOLS Fortran source files that are used in the AHA Model. Note that `BASE_STRINGS` in the list must go before `BASE_CSV_IO` because `BASE_CSV_IO` depends on procedures from `BASE_STRINGS`.
- **HEDTOOLSDIR** defines the path to the HEDTOOLS source code.
- **IEEEPATH** defines the path to the IEEE non-intrinsic math modules (part of HEDTOOLS).
- **WINRM** defines the command that is used to delete files and directories on the Windows platform. The native file delete command on the Windows platform is `del` or `erase`. However, if Cygwin is used, this native file removal command may not call with a "command not found" error. In such a case, use the Unix `rm` tool provided by Cygwin.

The source code of the `Makefile` contains many other parameters and is well documented throughout.

Chapter 3

Modern Fortran features

The AHA Model is implemented in the modern Fortran **F2003** and partially **F2008** standards. Because not all widespread compilers support the full set of these language standards, the model code uses only those that are supported by the following minimum compiler versions:

- GNU gfortran 4.8
- Intel Fortran 17

Modern Fortran fully supports object oriented programming with polymorphic objects and rich set of array functions which are also usable in the object oriented code.
This document provides a very brief overview of the main features of modern Fortran that are widely used in the AHA Model code.

3.1 Object oriented programming and type-bound procedures

Stated simply, the object-oriented programming paradigm is based on the notion of **object.** Here object is an entity that integrates **data** and **procedures** that are implemented to manipulate these data. In the simplest case, data can be considered as the "properties"" or "attributes" that describe the object. Procedures that are linked with the object, on the other hand, provide other derived attributes of the object or describe what the object can "do".
Different objects can be arranged in various ways (e.g. form more complex objects like arrays). For instance a population of agents (another object) can be simply formed by arranging individual agents (other objects) into an array. Various agents can also interact with each other.
For example, a single "agent" object is an entity having such attributes as sex, spatial position, body mass, body length etc. It can also have such boolean attributes as "is alive" (true or false). For any such object, one can calculate instantaneous risk of predation and other transient derived properties. Also, the agent can interact with objects of various other kinds. For example, an agent can change its spatial position (its position attribute is changed), approach a food item and "eat" it (basically, absorb the mass attribute of the item, the item is destroyed thereafter). Agent can also do many other things, e.g. "die". The functions that are linked to the object are usually called **methods.**
When an instance of the object is created, it is initialised in a function (e.g. `init`) that is often called the **constructor**. Another procedure is sometimes implemented to destroy and deallocate the object, it is the **destructor**.
Object-oriented code in modern Fortran is based on what is called **type-bound procedures**.
Briefly, a derived type is declared using the `type` keyword; it can contain several intrinsic and other derived types. Thus, a **data structure** is implemented.

```
type, public :: SPATIAL_POINT
  real(SRP) :: x, y, depth
  character(len=LABEL_LENGTH) :: label
  ....
end type SPATIAL_POINT
```

Any **instances** of the object can be declared using `type` keyword.

```
type(SPATIAL_POINT) :: some_point, another_point
```

A procedure can then be declared that operates specifically on this type. The first parameter `this` refers to the object that the procedure operates on.

The base object `this` is declared as `class` in the procedure, which allows to accept any **extension** of the `this` object as the first parameter. This is called "polymorphic objects."
Note that the other parameters (non `this`) can be declared as `class` or as `type`. In the former case, the procedure could accept any extensions (the procedure is then **polymorphic**) of the object, while in the latter, only this specific `type` (non-polymorphic procedure).
Components of the object are separated from its name with the percent sign `%`, e.g. the `x` coordinate is this%x.

```
function spatial_distance_3d (this, other) result (distance_euclidean)
  class(SPATIAL_POINT), intent(in) :: this
  real(SRP) :: distance_euclidean
  class(SPATIAL_POINT), intent(in) :: other
  distance_euclidean = dist( [this%x,  this%y,  this%depth],              &
                             [other%x, other%y, other%depth] )
end function spatial_distance_3d
```

The procedure is then included into the derived type declaration.
The name of the procedure that is implemented (e.g. `spatial_distance_3d` in the example above) is not called directly in calculations and can be declared `private`. Instead, a **public interface** name is declared in the derived type that defines how the procedure should be called, in the example below it is `distance`.
Note that the interface name can coincide for several different objects, however the actual procedure name (`spatial_distance_3d`) must be unique within the module that defines the derived type and its procedures.

```
type, public :: SPATIAL_POINT
  real(SRP) :: x, y, depth
  character(len=LABEL_LENGTH) :: label
  ....
  contains
  procedure, public :: distance => spatial_distance_3d
  ....
end type SPATIAL_POINT
```

Now, the procedure is called for the specific instance of the object (it comes to the procedure as the `this` first "self" parameter) using the public interface name (`distance`) rather than the "actual" procedure name (`spatial_↩ distance_3d`).

```
type(SPATIAL_POINT) :: point_a, point_b
...
distance_between_points = point_a%distance( point_b )
```

An **extension** object can be declared using `extends` keyword, that will use all the properties and type-bound procedures of the base object and add its own additional ones. This allows creating complex inheritance hierarchies across objects.

```
type, public, extends(SPATIAL_POINT) :: SPATIAL_MOVING
  ! The following component adds an array of history of the object
  ! movements:
  type(SPATIAL_POINT), dimension(HISTORY_SIZE_SPATIAL) :: history
  ...
  contains
    ....
    procedure, public :: go_up => spatial_moving_go_up
    procedure, public :: go_down => spatial_moving_go_down
    ....
end type SPATIAL_MOVING
```

Thus, the structure of the module that defines an inheritance hierarchy of objects and their type-bound functions is like this:

```
module SPATIAL_OBJECTS
  ! Declarations of objects:
  type, public :: SPATIAL_POINT
    real(SRP) :: x, y, depth
    character(len=LABEL_LENGTH) :: label
    ....
    contains
    procedure, public :: distance => spatial_distance_3d
    ....
  end type SPATIAL_POINT
  ....
  type, public, extends(SPATIAL_POINT) :: SPATIAL_MOVING
    ! The following component adds an array of history of the object
    ! movements:
    type(SPATIAL_POINT), dimension(HISTORY_SIZE_SPATIAL) :: history
```

```
    ...
    contains
      ....
      procedure, public :: go_up => spatial_moving_go_up
      procedure, public :: go_down => spatial_moving_go_down
      ....
  end type SPATIAL_MOVING
  .....
  ! other declarations
  .....
  contains
    ! Here go all the procedures declared in this module
    function spatial_distance_3d (this, other) result (distance_euclidean)
      class(SPATIAL_POINT), intent(in) :: this
      real(SRP) :: distance_euclidean
      class(SPATIAL_POINT), intent(in) :: other

      distance_euclidean = dist( [this%x,  this%y,  this%depth],              &
                                 [other%x, other%y, other%depth] )

    end function spatial_distance_3d
    ....
    ! Any other procedures
    ......
end module SPATIAL_OBJECTS
```

Relationships between different kinds of objects can be represented graphically in a class diagram. See Object-oriented programming and modelling section of the HEDTOOLS manual for more information.

3.2 Elemental procedures

Modern Fortran includes a powerful concept of **elemental procedures**. Such procedures (subroutines of functions) are declared using simple scalar parameters. However, they can also accept parameters that are arbitrary **arrays**. There are strict limits on the procedures that can be `elemental`. Basically, there should be no side effects in the procedure code (the code should only affect the defined parameters that are in the parameter list), if the code calls any procedures or functions, these should be declared as `pure` (free of any side effects), each parameter must have explicit `intent` declared. Notably, calling random numbers (`random_number`) or any input/output are not allowed.
Here is an example of a simple elemental function:

```
elemental function add (a,b) result (sum)
  real :: a, b, sum
  sum = a + b
end function add
```

It can be used with scalar arguments as normal:

```
real :: x, y, z
z = add(x, y)
```

However, it can also be used with arrays and combinations of arrays and scalars:

```
real :: a
real, dimension(100,100) :: x, y, z1, z2
z1 = add(a, x)          ! scalar added to array
z2 = add(x, y)          ! two arrays are added element by element
```

In such a case, the function is executed in element-wise manner. For this reason, the arrays should have conforming dimensionality and sizes.
Elemental procedures also work with the object oriented code. This allows implementation complex algorithm in a very concise manner, avoiding code duplication across different-sized arrays.
For example, the simple function below allows to get the spatial position of a spatial object or any extension.

```
elemental function spatial_get_current_pos_3d_o(this) result(coordinates)
  class(SPATIAL_POINT), intent(in) :: this
  type(SPATIAL_POINT) :: coordinates
  coordinates%x = this%x
  coordinates%y = this%y
  coordinates%depth = this%depth
end function spatial_get_current_pos_3d_o
```

However, it can also be used with an array of such spatial objects and returns an array of their positions. For example, it can return an array of spatial positions for a whole population of agents, or for its slice.

```
call LOG_DBG("Coordinates: " // TOSTR(parents%individual(1:10)%location()))
```

Using elemental procedures in the object oriented Fortran code allows to substitute such loop-based code:

```
do i = 1, size(neighbours)
  dist_here(i)  = this%distance(neighbours(i))
end do
```

by a shorter, simpler (and usually better optimised compiled code) whole-array based code:

```
dist_here  = this%distance(neighbours)
```

3.3 Whole array procedures

Most arithmetic functions in modern Fortran accept scalar as well as array arguments (usually elemental). There are also many other useful intrinsic array functions that allow writing very concise and easily understandable code. An additional advantage is that these procedures are usually highly optimised by the compiler and can be executed in parallel multi-threading mode if automatic parallelisation is enabled.

```
! Maximum and minimum value of an array:
a = maxval(array)
b = minval(array)

! ... and their locations:
i = maxloc(array)
j = minloc(array)

! array sum:
total = sum(array)

! count elements with a mask:
i = count( x > y )

! masked array assignment:
where ( x == missing ) x = y

! indexed parallel array assignment:
forall ( i=1:popsize ) x(i) = 1.0/i

! ... and many more
```

Using such whole-array based functions allows implementation of very concise and comprehensible code that is also highly optimised for multithreading execution.

```
! Number of agents alive
call csv_record_append(file_record, count(parents%individual%is_alive()))
....
! Average body mass
call csv_record_append(file_record, average(parents%individual%body_mass))
```

3.4 User defined operators

Modern Fortran allows to define arbitrary operators and redefine intrinsic operators.
User defined operators are declared using `interface operator` and must refer to a specific function that implements the operator.

```
interface operator (.above.)
  procedure spatial_check_located_below
end interface operator (.above.)
....
....
function spatial_check_located_below(this, check_object) result (are_below)
  class(SPATIAL_POINT), intent(in) :: this
  class(SPATIAL_POINT), intent(in) :: check_object
  logical :: are_below
  if ( check_object%dpos() > this%dpos() ) then
    are_below = .TRUE.
  else
    are_below = .FALSE.
  end if
end function spatial_check_located_below
```

Once defined, such operators can be used with object oriented code like the intrinsic operators:

```
do i=1, max
  if ( this%perceive_consp%conspecifics_seen(i) .above. the_agent ) then
    number_above = number_above + 1
  end if
end do
```

or using whole-array operators even simpler:

```
number_above=count(this%perceive_consp%conspecifics_seen .above. the_agent)
```

User-defined operators can refer to elemental functions, however can be used only with scalar arguments.

3.5 The associate construct

The `associate` construct allows to declare a shortcut that is then used in place of a long and complex expression or variable. This includes indexing variables in `do` loops and parts of the object hierarchy.
Here is an example:

```
do ind = 1, size(this%individual)
  ......
  associate ( AGENT => this%individual(ind) )
    call csv_record_append(file_record, AGENT%person_number        )
    call csv_record_append(file_record, AGENT%genome_label         )
    call csv_record_append(file_record, conv_l2r(AGENT%alive)      )
    call csv_record_append(file_record, conv_l2r(AGENT%sex_is_male))
    call csv_record_append(file_record, AGENT%body_length          )
    call csv_record_append(file_record, AGENT%body_length_birth    )
    call csv_record_append(file_record, AGENT%control_unselected   )
    call csv_record_append(file_record, AGENT%body_mass            )
    ....
  end associate
  .....
end do
```

3.6 Pointers

Modern Fortran supports **pointers,** in the same way as many other programming languages like C or C++. However, the use of pointers is more limited in Fortran, which is also not bad because inaccurate use of pointers can cause severe problems with the memory (e.g. when a pointer refers to a nonexistent object).
Basically, pointer is an alias to some other object, e.g. a variable or array. The use of pointers can make code more flexible and dynamic.
The pointer is declared as a specific type (intrinsic or derived) with the `pointer` attribute. The object that is used as the target for the pointer should be declared with the same type and the `target` attribute.

```
type(POPULATION), public, target ::  generation_one
type(POPULATION), public, pointer :: parents
```

After this, one can simply make the object `parents` an alias of `generation_one`:

```
parents => generation_one
```

Now the object `parents` points to the location in the computer memory where the object `generation_one` resides. It can consequently be used instead of the "target." For example, all the type-bound functions that are defined for `generation_one` can be called for `parents`.

```
call parents%sort_by_fitness()
```

3.7 References

- Adams, J. C., et al., (2009). The Fortran 2003 Handbook. Springer.
- Akin, E. (2003). Object-Oriented Programming via Fortran 90/95. Cambridge University Press.
- Brainerd, W. S. (2015). Guide to Fortran 2008 Programming. Springer.
- Chapman, S. J. (2007). Fortran 95/2003 for Scientists and Engineers. McGraw-Hill.
- Clerman, N. S., & Spector, W. (2012). Modern Fortran: Style and usage. Cambridge University Press.

Chapter 4

Modules Index

4.1 Modules List

Here is a list of all modules with brief descriptions:

Chapter 5

Data Type Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 6

Data Type Index

6.1 Data Types List

Here are the data types with brief descriptions:

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

Chapter 8

Module Documentation

8.1 commondata Module Reference

COMMONDATA – definitions of global constants and procedures.

Data Types

- type timer_cpu

  CPU timer container object for debugging and speed/performance control. Arbitrary timers can be instantiated for different parts of the code and also global. Using a specific timer (`stopwatch`) is like this:
- interface gamma2gene

  Sigmoidal relationship between environmental factor and the organism response, as affected by the genotype and environmental error, e.g. perception and neuronal response or intrinsic baseline and phenotypic hormone levels.
- interface gene2gamma
- interface add_to_history

  Simple history stack function, add to the end of the stack. We need only to add components on top (end) of the stack and retain `HISTORY_SIZE_SPATIAL` elements of the prior history (for a spatial moving object). The stack works as follows, assuming 100 and 200 are added:
  [1 2 3 4 5 6 7 8 9 10]
  *[2 3 4 5 6 7 8 9 10 **100**]*
  *[3 4 5 6 7 8 9 10 100 **200**].*
- interface cm2m

  Convert cm to m.
- interface m2cm

  Convert m to cm.
- interface mm2m

  Convert mm to m.
- interface rescale

  Arbitrary rescales value(s) from one range (A:B) to another (A1:B1).
- interface within

  Force a value within the range set by the vmin and vmax dummy parameter values.
- interface is_within

  *Logical function to check if a value is within a specific range, **lower** <= **X** <= **upper**.*
- interface is_near_zero

  Checks if a real number is near 0.0. Thus function can be used for comparing two real values like this:
- interface float_equal

  Check if two real values are nearly equal using the commondata::is_near_zero(). Thus function can be used for comparing two real values like this:
- interface operator(.feq.)

  "Float equality" operator: Check if two real values are nearly equal using the commondata::is_near_zero() function. Thus function can be used for comparing two real values like the below.

- interface operator(.approx.)

  "Approximatel equality" operator: Check if two real values are approximately equal using the commondata::is_near_zero() function. Thus function can be used for comparing two real values like the below.
- interface operator(.within.)

  Interface operators `.within.` for testing whether a value (first argument) lies within the limits set by a two-element array (second argument). All the values/parameters are Fortran intrinsic types, real or integer. Usage of the operator:
- interface operator(.cat.)

  Concatenate two arrays a and b. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.
- interface average

  Calculate an average of an array excluding missing code values.
- interface is_maxval

  Check if a value is the maximum value of an array.
- interface is_minval

  Check if a value is the minimum value of an array.
- interface operator(.radd.)

  Interface operator `.radd.` performs a random addition or subtraction of two numbers with equal probability. See `commondata::random_add_subtract()`. The operator can be used as follows:

Functions/Subroutines

- elemental real(srp) function, private cm2m_r (value_cm)

  Convert cm to m.
- elemental real(hrp) function, private cm2m_hr (value_cm)

  Convert cm to m.
- elemental real(srp) function, private cm2m_i (value_cm)

  Convert cm to m.
- elemental real(srp) function, private m2cm_r (value_m)

  Convert m to cm.
- elemental real(hrp) function, private m2cm_hr (value_m)

  Convert m to cm.
- elemental real(srp) function, private m2cm_i (value_m)

  Convert m to cm.
- elemental real(srp) function, private mm2m_r (value_mm)

  Convert mm to m.
- elemental real(srp) function, private mm2m_i (value_mm)

  Convert mm to m.
- elemental real(srp) function carea (R)

  Calculate a circle area.
- elemental real(srp) function length2sidearea_fish (body_length)

  *A function linking **body length** with the body **area** in fish.*
- elemental real(srp) function, private rescale_full (value_in, A, B, A1, B1)

  Rescale a real variable with the range A:B to have the new range A1:B1.
- elemental real(srp) function, private rescale_1 (value_in, A1, B1)

  Rescale a real variable with the range 0:1 to have the new range A1:B1.
- elemental real(srp) function, private within_r (value_in, vmin, vmax)

  Force a value within the range set by the vmin and vmax dummy parameter values. If the value is within the range, it does not change, if it falls outside, the output force value is obtained as min(max(value, FORCE_MIN), FORCE_MAX)
- elemental integer function, private within_i (value_in, vmin, vmax)

Force a value within the range set by the vmin and vmax dummy parameter values. If the value is within the range, it does not change, if it falls outside, the output force value is obtained as min(max(value, FORCE_MIN), FORCE_MAX)

- elemental logical function, private is_within_r (x, lower, upper)

  Logical function to check if a value is within a specific range, ***lower <= X <= upper****. The reverse (upper <= x <= lower) range limits can also be used; a corrective adjustment is automatically made.*
- elemental logical function, private is_within_i (x, lower, upper)

  Logical function to check if a value is within a specific range, ***lower <= X <= upper****. The reverse (upper <= x <= lower) range limits can also be used; a corrective adjustment is automatically made.*
- pure logical function, private is_within_operator_r (x, limits)

  A wrapper function for commondata::is_within() to build a user defined operator. Basically, it is the same as `is_within`*, but the lower and upper limits are set as a two-element array. Usage of the operator:*
- pure logical function, private is_within_operator_i (x, limits)

  A wrapper function for commondata::is_within() to build a user defined operator. Basically, it is the same as `is_within`*, but the lower and upper limits are set as a two-element array. Usage of the operator:*
- pure real(srp) function, private average_r (array_in, missing_code, undef_ret_null)

  Calculate an average value of a real array, excluding MISSING values.
- pure real(srp) function, private average_i (array_in, missing_code, undef_ret_null)

  Calculate an average value of an integer array, excluding MISSING values.
- real(srp) function std_dev (array_in, missing_code, undef_ret_null)

  Calculate standard deviation using trivial formula:
- pure real(srp) function, dimension(:), allocatable, private stack2arrays_r (a, b)

  Concatenate two arrays a and b. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` *method.*
- pure integer function, dimension(:), allocatable, private stack2arrays_i (a, b)

  Concatenate two arrays a and b. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` *method.*
- elemental logical function, private is_near_zero_srp (test_number, epsilon)

  Checks if a real number is near 0.0. Thus function can be used for comparing two real values like the below.
- elemental logical function, private is_near_zero_hrp (test_number, epsilon)

  Checks if a real number is near 0.0. Thus function can be used for comparing two real values like the below.
- elemental logical function, private float_equal_srp (value1, value2, epsilon)

  Check if two real values are nearly equal using the commondata::is_near_zero(). Thus function can be used for comparing two real values like the below. The exact comparison (incorrect due to possible rounding):
- elemental logical function, private float_equal_hrp (value1, value2, epsilon)

  Check if two real values are nearly equal using the commondata::is_near_zero(). Thus function can be used for comparing two real values like the below. The exact comparison (incorrect due to possible rounding):
- elemental logical function, private float_equal_srp_operator (value1, value2)

  This is a wrapper for the commondata::float_equal_srp() for building the user defined operator `.feq.` *with default tolerance (*`epsilon` *parameter). The exact real comparison (incorrect due to possible rounding):*
- elemental logical function, private float_equal_hrp_operator (value1, value2)

  This is a wrapper for the commondata::float_equal_hrp() for building the user defined operator `.feq.` *with default tolerance (*`epsilon` *parameter). The exact real comparison (incorrect due to possible rounding):*
- elemental logical function, private float_approx_srp_operator (value1, value2)

  This is a wrapper for the commondata::float_equal_srp() for building the user defined operator `.approx.` *with* ***very high*** *tolerance (*`epsilon` *parameter). The exact real comparison (incorrect due to possible rounding):*
- elemental logical function, private float_approx_hrp_operator (value1, value2)

  This is a wrapper for the commondata::float_equal_hrp() for building the user defined operator `.approx.` *with* ***very high*** *tolerance (*`epsilon` *parameter). The exact real comparison (incorrect due to possible rounding):*
- subroutine do_sanitise (array, lvalid, hvalid, substval, only_wrong, tnote)

  Sanitize a real commondata::srp array, so that any value that is smaller than the minimum sensible value `lvalid` *or greater then the maximum sensible value* `hvalid` *is substituted with* `substval`*. The procedure also checks the input value for IEEE validity: overflow, underflow, invalid and inexact.*
- integer function ieee_error_reporting (reset, tnote)

Check if an IEEE error condition has occurred.

- real(srp) function zeroin (ax, bx, f, tol)

  This function calculates a zero of a function f(x) in the interval (ax,bx).
- elemental real(srp) function allelescale (raw_value)

  Converts and rescales integer allele value to real value for neural response function.
- elemental real(srp) function alleleconv (raw_value)
- elemental real(srp) function cv2variance (cv, mean)

  Calculate the variance from the coefficient of variation.
- real(srp) function, private gamma2gene_additive_i4 (gs, gh, signal, erpcv)

  The function gamma2gene finds the sigmoid relationship for a complex multicomponent 2-allele impact on the neuronal response.
- real(srp) function, private gamma2gene_additive_r4 (gs, gh, signal, erpcv)

  The function gamma2gene finds the sigmoid relationship for a complex multicomponent 2-allele impact on the neuronal response.
- elemental real(srp) function, private gamma2gene_fake_vals (signal, gs, gh, n_acomps)

  This "fake" version of the `gamma2gene` is used to guess the response values in calculations.
- elemental real(srp) function, private gamma2gene_reverse (neuronal_response, gs, gh, nc)

  Reverse-calculate perception value from the given neural response value.
- pure subroutine, private add_to_history_i4 (history_array, add_this)

  Simple history stack function, add to the end of the stack. We need only to add components on top of the stack and retain `commondata::history_size_spatial` elements of the prior history (for a spatial moving object). The stack works as follows, assuming 100 and 200 are added:
  [1 2 3 4 5 6 7 8 9 10];
  [2 3 4 5 6 7 8 9 10 100];
  [3 4 5 6 7 8 9 10 100 200].
- pure subroutine, private add_to_history_r (history_array, add_this)

  Simple history stack function, add to the end of the stack. We need only to add components on top of the stack and retain `commondata::history_size_spatial` elements of the prior history (for a spatial moving object).
- pure subroutine, private add_to_history_char (history_array, add_this)

  Simple history stack function, add to the end of the stack. We need only to add components on top of the stack and retain `commondata::history_size_spatial` elements of the prior history.
- elemental integer function conv_l2i (flag, code_false, code_true)

  Converts logical to integer following a rule, default FALSE = 0, TRUE = 1.
- elemental real(srp) function conv_l2r (flag, code_false, code_true)

  Converts logical to standard (kind SRP) real, .FALSE. => 0, .TRUE. => 1.
- pure logical function, private is_maxval_r (value, array, tolerance)

  Function to check if the value is the maximum value of an array (returns TRUE), or not (return FALSE).
- pure logical function, private is_maxval_i (value, array)

  Function to check if the value is the maximum value of an array (returns TRUE), or not (return FALSE). Integer version.
- pure logical function, private is_minval_r (value, array, tolerance)

  Function to check if the value is the minimum value of an array (returns TRUE), or not (return FALSE).
- pure logical function, private is_minval_i (value, array)

  Function to check if the value is the minimum value of an array (returns TRUE), or not (return FALSE). Integer version.
- subroutine, private timer_cpu_start (this, timer_title)

  Start the timer object, stopwatch is now ON.
- real(srp) function, private timer_cpu_elapsed (this)

  Calculate the time elapsed since the stopwatch subroutine was called for this instance of the timer container object. Can be called several times showing elapsed time since the grand start.
- character(len=:) function, allocatable, private timer_cpu_title (this)

  Return the title of the current timer object.
- character(len=:) function, allocatable, private timer_cpu_show (this)

  A ready to use in output function that returns a formatted string for a timer combining its title and the elapsed time. For example: `Calculating decomposition took 20s`.

- character(len=:) function, allocatable, private timer_cpu_log (this)

  A ready to use shortcut function to be used in logger, just adds the TIMER: tag in front of the normal `show`*output.* ***Example use:***
- subroutine call_external (command, suppress_output, suppress_error, is_background_task, cmd_is_↩ success, exit_code)

  Call an external program using a command line. Wrapper to two alternative system shell calling intrinsic procedures.
- logical function check_external (exec)

  Check if an external procedure is executable and can be run.
- subroutine log_check_external (exec, debug_only, is_valid)

  Check if an external procedure can be called and log the result.
- subroutine debug_histogram_save (x_data, delete_csv, csv_out_file, enable_non_debug)

  Produce a ***debug plot*** *of histogram using an external program* `hthist` *from HEDTOOLS tools.*
- subroutine debug_scatterplot_save (x_data, y_data, delete_csv, csv_out_file, enable_non_debug)

  Produce a ***debug plot*** *of 2-d scatterplot using an external program* `htscatter` *from HEDTOOLS tools.*
- subroutine debug_interpolate_plot_save (grid_xx, grid_yy, ipol_value, algstr, output_file, enable_non_debug)

  Produce a ***debug plot*** *of the* ***interpolation data*** *using an external program* `htinterp` *from the HEDTOOLS tools.*
- subroutine file_delete (file_name, success)

  Delete a file from the local file system using Fortran open status=delete or fast POSIX C call.
- real(srp) function, private random_add_subtract (x, y)

  Random operator, adds or subtracts two values with equal probability, used in the random walk functions.
- subroutine system_init ()

  Initialises the system environment and sets basic parameters.
- subroutine system_halt (is_error, message, ignore_lockfile)

  Halt execution of the system with a specific message and exit code. The exit code is normally passed to the operating system. However, this behaviour is implementation dependent and can be unexpected on specific the platform(s) and the compiler(s).
- subroutine, private logger_init ()

  logger_init *Initialise the system and the system logger.*
- subroutine log_dbg (message_string, procname, modname)

  `LOG_DBG`*: debug message to the log. The message goes to the logger only when running in the DEBUG mode.*
- subroutine log_ieee (ttag, always_log, reset_flags)

  `LOG_IEEE`*: Check and log IEEE signalling flags. Logging normally occurs only if any nonzero output from ieee_error_reporting() is found.*
- character(len=:) function, allocatable, private parse_svn_version ()

  Parse and cut revision ***number*** *in form of string from the whole SVN revision string. SVN revision number can therefore be included into the model outputs and output file names. This is convenient because the model version is identified by a single SVN revision number.*
- character(len=long_label_length) function, dimension(:), allocatable parse_abstract (file_name)

  Get and parse the model abstract. Model abstract is a short descriptive text that can span several lines and is kept in a separate file that is defined by the commondata::model_abstract_file.
- character(len=:) function, allocatable, private tag_mmdd ()

  Date (YYYYMMDD) tag for file names and logs.

Variables

Precision control for real type and IEEE float math in the model

- integer, parameter, public s_prec_32 = selected_real_kind(6, 37)

  Standard precision for real data type. We first define 32, 64 and 128 bit real kinds.
- integer, parameter, public d_prec_64 = selected_real_kind(15, 307)
- integer, parameter, public q_prec_128 = selected_real_kind(33, 4931)
- integer, parameter, public srp = S_PREC_32

  Definition of the ***standard*** *real type precision (SRP).*
- integer, parameter, public hrp = Q_PREC_128

*Definition of the **high** real precision (**HRP**). This real type kind is used in pieces where a higher level of FPU precision is required, e.g. to avoid overflow/underflow and similar errors.*

- integer, parameter, public long = selected_int_kind(16)

  In some (perhaps quite rare) cases of exponentiation we may also need huge integers, those in 64 bit would probably be enough. So whenever we need such a big integer, declare it as:

Accessory parameters

- character(len= ∗), parameter, private modname = "(COMMONDATA)"

  MODNAME always refers to the name of the current module for use by the LOGGER function LOG_DBG. Note that in the debug mode (if IS_DEBUG=TRUE) LOGGER should normally produce additional messages that are helpful for debugging and locating possible sources of errors. MODNAME is declared private and is not accessible outside of this module. Each procedure should also have a similar private constant `commondata::procname`.
- character(len= ∗), parameter, private procname = ""

  PROCNAME is the procedure name for logging and debugging (with `commondata::modname`).
- character(len= ∗), parameter, public svn_version_string = "$Revision: 9552 $"

  ***Subversion** or Mercurial revision number (or ID) of the model code.*
- character(len=:), allocatable, public, protected svn_version

  ***Subversion** or Mercurial revision number that is parsed by `commondata::parse_svn_version()`. It is shorter than `commondata::svn_version_string` and does not contain blanks. Therefore, it can be used for building output file names.*
- logical, parameter, public true =.TRUE.

  Safety parameter avoid errors in logical values, so we can now refer to standard Fortran `.TRUE.` and `.FALSE.` as `YES` and `NO` or `TRUE` and `FALSE`
- logical, parameter, public false =.FALSE.
- logical, parameter, public yes =.TRUE.
- logical, parameter, public no =.FALSE.
- real(srp), parameter, public zero = epsilon(0.0_SRP)

  Some parameters should never be zero or below. In such cases they could be set to some smallest distinguishable non-zero value. Here set as the Fortran intrinsic `epsilon` function, a value that is almost negligible compared to one, i.e. the smallest `real` number E such that $1 + E > 1$. In some cases it is also reasonable to set the tolerance limit to this parameter (see Float point computations).
- real(srp), parameter, public tiny_srp =tiny(1.0_SRP)

  The smallest positive number in the commondata::srp standard real model.
- real(hrp), parameter, public tiny_hrp =tiny(1.0_HRP)

  The smallest positive number in the commondata::hrp high precision real model. See Float point computations.
- real(srp), parameter, public lo_valid_sanitised = TINY_SRP ∗ 10.0_SRP

  Lower bound for do_sanitise() procedure. This is the lowest value that considered valid.
- real(srp), parameter, public hi_valid_sanitised = huge(1.0_SRP)/100.0_SRP

  Higher bound for do_sanitise() procedure. This is the highest value that considered valid.
- real(srp), parameter, public tolerance_low_def_srp = TINY_SRP ∗ 5.0_SRP

  Default value of low tolerance (high precision). This is the standard commondata::srp precision. See Float point computations.
- real(hrp), parameter, public tolerance_low_def_hrp = TINY_HRP ∗ 5.0_HRP

  Default value of low tolerance (high precision). This is the high commondata::hrp precision. See Float point computations.
- real(srp), parameter, public tolerance_high_def_srp = ZERO ∗ 1000.0_SRP

  Default value of high tolerance (low precision). This is the standard commondata::srp precision real. See Float point computations.
- real(hrp), parameter, public tolerance_high_def_hrp = epsilon(0.0_HRP) ∗ 1000.0_HRP

  Default value of high tolerance (low precision). This is the high commondata::hrp precision real. See Float point computations.
- real(srp), parameter, public missing = -9999.0_SRP

  *Numerical code for missing and invalid **real type** values.*
- real(srp), parameter, public invalid = -9999.0_SRP
- integer, parameter, public unknown = -9999

  *Numerical code for invalid or missing **integer** counts.*
- real(srp), parameter, public pi =4.0_SRP∗atan(1.0_SRP)

  *The **PI** number.*
- character(len= ∗), parameter, public csv =".csv"

  Standard data file extension for data output is now .csv.

- character(len= ∗), parameter, public ps =".ps"

  Standard file extension for debug and other PostScript plots.
- integer, parameter, public filename_length = 255

  Set the standard length of the file name, are 255 characters enough?
- logical, parameter, public use_posix_fs_utils = .TRUE.

  Logical flag for setting if POSIX direct filesystem procedures are used. These utilities are implemented in HEDTOOLS for standard POSIX C call via the Fortran interface. They should work safer, better and faster than indirect procedure wrappers (e.g. calling `system()`) but are not fully portable and might not work as expected on all systems and compilers.
- integer, parameter, public label_length = 14

  The length of standard character string labels. We use labels for various objects, like alleles, perceptual and neural components / bundles etc. For simplicity, they all have the same length. It should be big enough to fit the longest whole label.
- integer, parameter, public long_label_length = 128

  The length of long labels.
- integer, parameter, public label_cst = 97

  This parameter defines the range of characters that is used for generating random labels, 97:122 corresponds to lowercase Latin letters.
- integer, parameter, public label_cen = 122
- character(len= ∗), parameter lock_file = "lock_simulation_running.lock"

  The name of the lock file. The lock file is created at the start of the simulation and is deleted at the end of the simulation. It can be used to signal that simulation is still ongoing to external utilities and scripts. See The lock file.
- integer, public, protected lock_file_unit

  This is the unit number that identifies the lock file. The lock file is created at the start of the simulation and is deleted at the end of the simulation. It can be used to signal that simulation is still ongoing to external utilities and scripts. See The lock file.
- character(len= ∗), parameter stop_file = "stop_simulation_running.lock"

  The name of the stop file. The stop file is checked before each new generation of the Genetic Algorithm. If this file is found, simulation does not go to the next generation and just stops. See The stop file.
- integer, parameter, public platform_windows = 100

  Runtime platform ID constants. Use these constants for determining the current runtime platform, e.g. `Platform_Running = PLATFORM_WINDOWS`. See `commondata::platform_running`.
- integer, parameter, public platform_unix = 111
- integer, public platform_running

  Global variable that shows what is the current platform. Should use the above platform constants, e.g. `Platform_Running = PLATFORM_WINDOWS`. See `commondata::platform_windows` and `commondata::platform_unix`.
- character(len= ∗), parameter, public exec_interpolate = "htintrpl.exe"

  *There are a few **external programs** which are called from the model code. The name of the **interpolation** program (htintrpl.f90 from HEDTOOLS) executable.*
- character(len= ∗), parameter, public exec_scatterplot = "htscatter.exe"

  *The name of the **scatterplot** program (htscatter.f90 from HEDTOOLS) executable.*
- character(len= ∗), parameter, public exec_histogram = "hthist.exe"

  *The name of the **histogram** program (hthist.f90 from HEDTOOLS) executable.*
- character(len= ∗), parameter, public ltag_major = "IMPORTANT: "

  ***Tag prefixes** for the logger system. The log may use tags for some common information pieces, so they are easily found within. The tags are normally set the prefix for the log: 017-01-31 13:33:22 INFO: Saving histogram, data: debug_hist.csv Some common tags are: STAGE STAGE: 2017-01-31 16:03:15 INFO: Generation 7 took 448.3279s. INFO INFO: some information TIMER TIMER: Calculating distances took 0.001 s Tag meaning:*

- character(len= ∗), parameter, public ltag_stage = "STAGE: "
- character(len= ∗), parameter, public ltag_info = "INFO: "
- character(len= ∗), parameter, public ltag_warn = "WARNING: "
- character(len= ∗), parameter, public ltag_error = "ERROR: "
- character(len= ∗), parameter, public ltag_crit = "CRITICAL: "
- character(len= ∗), parameter, public ltag_timer = "TIMER: "
- character(len= ∗), parameter, public ltag_stats = "STATS: "

System-wide fatal errors

The description of errors that pertain to the whole system.

- character(len= ∗), parameter, public error_no_autoalloc = "No automatic array allocation"

Error message for ∗∗"no automatic intrinsic array allocation"∗∗. Fortran compilers support automatic allocation of arrays on intrinsic assignment. This feature should work by default in GNU gfortran v.4.6 and Intel ifort v.17.↩ 0.1. Automatic allocation allows to avoid a possible bug when the number of array elements in the `allocate` statement is not updated when the components of the array are updated in the array constructor.

- character(len= ∗), parameter, public error_auto_param_arrays = "No automatic size in parameter arrays"

  Error message for ∗∗"no automatic determination of the size in parameter"∗∗ arrays in the style:
- character(len= ∗), parameter, public error_allocation_fail = "Cannot allocate array or object"

  Error message ∗∗"Cannot allocate array or object"∗∗ is issued if an array or an object is checked and turns out to be not allocated while it must be.
- character(len= ∗), parameter, public error_lock_preexists = "Lock file '" // LOCK_FILE // "' exists. Is another simulation running?"

General Parameters

- character(len= ∗), parameter, public model_name = "HEDG2_04"

  Model name for tags, file names etc. Must be very short. See Model descriptors.
- character(len= ∗), parameter, public model_descr = "AHA, single fear, body size non-genetic."

  Model description - a fixed descriptive text, used in text outputs etc. See Model descriptors.
- character(len= ∗), parameter, private model_abstract_file = "abstract.txt"

  The name of the file that contains the Model abstract, a short description that can span several lines of text ans is kept in a separate file. The file is read, if it exists, and its contents is logged at the start the simulation. The separate Model Abstract file is useful because it can integrate dynamic information, such as the latest version control log(s) via Subversion or Mercurial hooks mechanism. See Model descriptors.
- logical, public, protected is_debug =.FALSE.

  Sets the model in the debug mode if TRUE. The Debug mode generates huge additional outputs and logs. Also, the logs by default go to the screen (standard output). See `commondata::system_init()` for details.
- logical, public, protected is_plotting =.TRUE.

  This parameter controls if the debug plots are produced. They can be huge number that takes lots of space. Also, debug plots are called as separate processes that can run at the background and easily exceed the system-specific limit on child processes (if run in asynchronous mode). Generation of debug plots can be controlled by the environment variable `AHA_DEBUG_PLOTS`: if it is set to TRUE, 1, or YES, debug plots are enabled. See `commondata::system_init()` for details.
- logical, public, protected is_screen_output =.FALSE.

  Sets the model in screen output mode. If TRUE, the logger output goes to the screen (standard output device). Can be manipulated using the environment variable `AHA_SCREEN`. If `AHA_SCREEN` is set to TRUE or 1 or yes, logger screen output is enabled. See `commondata::system_init()` for details.
- logical, public, protected is_zip_outputs =.FALSE.

  This parameter enables or disables post-processing compression of the data: if TRUE, the data are compressed using the command defined by the commondata::cmd_zip_output string parameter. Note that not all data files are compressed, only potentially big ones are (e.g. agent population data and habitat data).
- logical, parameter, public zip_outputs_background =.TRUE.

  This parameter defines if the output files are compressed in the background in the parallel mode or the program should wait for termination of the child zipping process.
- character(len= ∗), parameter, public cmd_zip_output = "gzip"

  This parameter defines the compression program that is executed to "zip" the data files if commondata::is_zip_outputs is enabled (TRUE). The normal compression utility is "gzip," that is found on almost any Linux/Unix system. gzip compresses each file individually and by default automatically deletes the original file. The compressed file extension is defined by commondata::zip_file_extenssion. See `http://www.gzip.org/`. Alternative compressors that are fairly widespread are `bzip2`, `lzma` and `xz`.
- character(len= ∗), parameter, public zip_file_extenssion = ".gz"

  This parameter defines the compressed file extension for the external compression utility defined by the commondata::cmd_zip_output.
- logical, parameter, public enable_save_agents_each_timestep = .FALSE.

  This parameter defines if all agents data is saved at each time step of the life cycle. See the_evolution::lifecycle↩ _preevol().
- character(len=:), allocatable, public, protected mmdd

  `MMDD` tag, year, month and day, used in file names and outputs. The value of the tag should be obtained only once at the start of the simulation, normally by calling the commondata::tag_mmdd() function at commondata::system_init(). It does not make much sense to generate these data tags on the fly as the simulations can be very long, several days, and so the file tags will be inconsistent.
- integer, parameter, public popsize = 10000

Maximum population size.

- integer, parameter, public generations = 100

  Maximum number of generations in GA.
- integer, public global_generation_number_current

  *The current global **generation number**. This is a global non fixed-parameter variable that is updated in subroutines.*
- integer, parameter, public lifespan = 14000

  Number of time steps in the agent's maximum life length.
- integer, parameter, public preevol_tsteps = 560

  Number of time steps in the agent's life at the pre-evolution stage.
- integer, parameter, public preevol_tsteps_force_debug = 280

  *Number of time steps in the agent's life at the fixed fitness pre-evolution stage. This parameter **forces** a smaller fixed value that is used for debugging only. Thus, adaptive time steps calculated by the_evolution::preevol_steps_adaptive() are disabled. To enable this fixed time steps, set this parameter commondata::preevol_tsteps_force_debug_enabled to TRUE.*
- logical, parameter, public preevol_tsteps_force_debug_enabled = .FALSE.

  This parameter enables the forced smaller fixed number of time steps set by the commondata::preevol_tsteps_force_debug parameter.
- logical, parameter, public lifecycle_predation_disabled_debug = .FALSE.

  This parameter completely disables predation in the GA life cycle procedure.
- integer, public global_time_step_model_current

  *The current global **time step** of the model. This is a global non fixed-parameter variable that is updated in subroutines.*
- integer, public global_frame_number

  The current global time frame. Frames are time steps within the time step defined by the commondata::global_time_step_model_curren
- real(srp), parameter, public percept_error_cv_def = 0.01_SRP

  Default perception error in the commondata::gamma2gene() neuronal response functions. Note that this parameter defines stochastic error as the Coefficient of Variation (CV).

Basic agent parameters

- real(srp), parameter, public body_length_min = 0.2_SRP

  Minimum body length possible.
- real(srp), parameter, public body_length_max = 100.0_SRP

  Maximum body length.
- real(srp), parameter, public body_mass_min = 0.1_SRP

  Minimum possible body mass, hard limit.
- logical, parameter, public init_agents_depth_is_fixed = .FALSE.

  This parameter determines if the agents are initialised at a fixed depth at the initialisation. Agents are normally placed uniformly, the_environment::uniform(), at the initialisation. However, the depth can be fixed. In such a case they are scattered uniformly in the X and Y coordinates but with fixed depth that is set by the commondata::init_agents_depth parameter.
- logical, parameter, public init_agents_depth_is_gauss = .TRUE.

  This parameter determines if the agents are initialised at a fixed depth at the initialisation. Agents are placed uniformly, the_environment::uniform(), at the initialisation. However, the depth can be a Gaussian value with the.
- real(srp), parameter, public init_agents_depth = 1833.0_SRP

  The fixed depth at which the agents are initialised at the start of the simulation. The other coordinates are normally set the_environment::uniform() within the initialisation environment container. See the_population::member_population::place_uniform().
- real(srp), parameter, public init_agents_depth_cv = 0.2_SRP

  This parameter sets the Coefficient of Variation for the Gaussian depth initialisation of the agents that is controlled by commondata::init_agents_depth_is_gauss. See the_population::member_population::place_uniform().
- real(srp), parameter, public reproduction_cost_body_mass_fix = 0.2_SRP

  The energetic cost of reproduction in terms of the agent's body mass loss.
- real(srp), parameter, public reproduction_cost_offspring_fract_male = 0.3_SRP

  *The component of the energetic cost of reproduction in **males** that is proportional to the total offspring mass. For details see the procedure `the_body::reproduction_cost_energy_dynamic()`.*
- real(srp), parameter, public reproduction_cost_offspring_fract_female = 1.0_SRP

  *The component of the energetic cost of reproduction in **females** that is proportional to the total offspring mass. For details see the procedure `the_body::reproduction_cost_energy_dynamic()`.*
- real(srp), parameter, public reproduction_cost_body_mass_factor_male = 0.4_SRP

*The component of the energetic cost of reproduction in **males** that is proportional to the agent's body mass. For details see the procedure `the_body::reproduction_cost_energy_dynamic()`.*

- real(srp), parameter, public reproduction_cost_body_mass_factor_female = 0.1

  *The component of the energetic cost of reproduction in **females** that is proportional to the agent's body mass. For details see the procedure `the_body::reproduction_cost_energy_dynamic()`.*
- real(srp), parameter, public reproduction_cost_unsuccess = 0.1_SRP

  *The energetic cost of unsuccessful reproduction in terms of the agent's body mass lost. This is a fraction of the **full cost of reproduction**, that is described by the `REPRODUCTION_COST_BODY_MASS` parameter.*
- real(srp), dimension(∗), parameter, public reproduct_body_mass_offspr_abscissa = [BODY_MASS_MIN, 3.0_SRP, 10.5_SRP, 12.0_SRP]

  *The array defining the **abscissa** (X) of the nonparametric function curve that defines the relationship between the agent's body mass and the overall mass of all offspring as a fraction of the agent's body mass.*
- real(srp), dimension(∗), parameter, public reproduct_body_mass_offspr_ordinate = [0.0_SRP, 0.1_SRP, 0.199_SRP, 0.20_SRP]

  *The array defining the **ordinate** (Y) of the nonparametric function curve that defines the relationship between the agent's body mass and the overall mass of all offspring as a fraction of the agent's body mass. Plotting command for the interpolator:*

Parameters of the environment

- real(srp), dimension(3), parameter, public universe_min_coord_notuse = [0.0_SRP, 0.0_SRP, 0.0_SRP]

  Overall size of the global 3D universe of the model.
- real(srp), dimension(3), parameter, public universe_whole_size_notuse = [20000.0_SRP, 10000.0_SRP, 3000.0_SRP]
- integer, parameter, public dielcycles = 100

  Number of days and nights in a lifespan, `DIELCYCLES=500`.
- integer, parameter, public history_size_spatial = 50

  The size of the history for spatial moving objects, i.e. how many time steps positions to remember in stack arrays.
- real(srp), dimension(3), parameter, public habitat_safe_min_coord = [0.0_SRP, 0.0_SRP, 0.0_SRP]

  Definition of the habitat spatial limits.
- real(srp), dimension(3), parameter, public habitat_safe_max_coord = [10000.0_SRP, 10000.0_SRP, 3000.0_SRP]
- real(srp), dimension(3), parameter, public habitat_danger_min_coord = [10000.0_SRP, 0.0_SRP, 0.0_↩ SRP]
- real(srp), dimension(3), parameter, public habitat_danger_max_coord = [20000.0_SRP, 10000.0_SRP, 3000.0_SRP]
- integer, parameter, public predators_num_habitat_safe = 100

  *The **number of predators** in the **safe** habitat.*
- integer, parameter, public predators_num_habitat_danger = 500

  *The **number of predators** in the **dangerous** habitat.*
- integer, parameter, public food_abundance_habitat_safe = 20000

  *The **food abundance** in the **safe** habitat.*
- integer, parameter, public food_abundance_habitat_danger = 40000

  *The **food abundance** in the **dangerous** habitat.*
- real(srp), parameter, public other_risks_def = 0.01_SRP

  Default level of other mortality risks in the habitat.
- real(srp), parameter, public other_risks_habitat_safe = 0.01_SRP

  *Habitat-specific mortality risk (not linked with predation) in the **safe** habitat.*
- real(srp), parameter, public other_risks_habitat_danger = 0.05_SRP

  *Habitat-specific mortality risk (not linked with predation) in the **dangerous** habitat.*
- real(srp), parameter, public eggmortality_def = 0.01_SRP

  Default level of egg mortality in the habitat.
- real(srp), parameter, public individual_mortality_risk_def = 0.01_SRP

  Default individually-specific mortality risk. It can increase or decrease depending on various factors. The individually-specific mortality risk is normally a Gaussian variable with the variability set by the commondata::individual_mortality_risk_cv.
- real(srp), parameter, public individual_mortality_risk_cv = 0.05_SRP

  The coefficient of variation for Gaussian stochastic individually-specific mortality risk of the agent.
- real(srp), parameter, public predator_body_size = 100.0_SRP

The body size of the predator. In this version all predators have the same body size set by this parameter, but can be Gaussian stochastic. Moreover, in such a case predator attack efficiency can depend on the body size, e.g. larger predators are more dangerous. compare to the agents maximum body size `BODY_LENGTH_MAX=100.0`

- real(srp), parameter, public predator_attack_rate_default = 0.9_SRP

  Mean rate of a single predator attack.
- real(srp), parameter, public predator_attack_rate_cv = 0.1_SRP

  Coefficient of variation for a single predator attack among the whole population of stochastic predators.
- real(srp), parameter, public predator_attack_capture_probability_half = 0.8_SRP

  The probability of capture of a fish agent by a predator at the distance equal to 1/2 of the visual range. For more details see `the_environment::predator_capture_risk_calculate_fish()`.
- real(srp), parameter, public predator_attack_capture_probability_min = 0.1_SRP

  Minimum probability of capture, e.g. at a distance exceeding the visual range. The latter assumes that the predator could detect the agent beyond the visual range and pursue it. For more details see `the_environment::predator_capture_risk_calculate_fish()`.
- real(srp), parameter, public predator_attack_capture_prob_frz_50 = 0.10_SRP

  A parameter factor defining the probability of capture of an immobile (freezing) agent by a predator: interpolation ordinate for the distance equal to ***0.25 of the visual range****. See* `the_environment::predator_capture_risk_calculate` *for details.*
- real(srp), parameter, public predator_attack_capture_prob_frz_75 = 0.01_SRP

  A parameter factor defining the probability of capture of an immobile (freezing) agent by a predator: interpolation ordinate for the distance equal to ***0.40 of the visual range****. See* `the_environment::predator_capture_risk_calculate` *for details.*
- logical, parameter, public agent_can_assess_predator_attack_rate = .TRUE.

  A logical flag of whether the agents can assess the individual inherent attack rates of the predators. If yes, these inherent individual attack rates are collated into the perception object. If no, the default attack rate set by the commondata::predator_attack_rate_default parameter is used.
- integer, parameter, public predator_risk_group_select_index_partial = 20

  Sets the limit for partial indexing and ranking of ***prey agents*** *in the visual range of the predator. The risk of predation, i.e. the probability of attack and capture of each agent in a group of agents, will be calculated individually for distance-ranked agents only up to this parameter value.*
- real(srp), dimension(∗), parameter, public predator_risk_group_dilution_ordinate = [1.0_SRP, 0.3_SRP, 0.1_SRP]

  The array defining the ordinate grid values for the weighting nonparametric function linking the distance rank of the agent within the visual field of the predator and the weighting factor adjusting for predator confusion and predator dilution effects. The grid abscissa is calculated dynamically in the the_environment::predator_capture_risk_calculate_fish_group() procedure.
- real(srp), parameter, public food_item_size_default = 2.1_SRP

  Default size of a single food item.
- real(srp), parameter, public food_item_mean_size = FOOD_ITEM_SIZE_DEFAULT

  The above is also the average size of a stochastic Gaussian food items.
- real(srp), parameter, public food_item_size_default_cv = 0.1_SRP

  Coefficient of variation for Gaussian food items.
- real(srp), parameter, public food_item_minimum_size = 1.0_SRP

  The minimum size of a food item. This is the "floor" in case the stochastically generated (e.g. Gaussian) value gets zero or below.
- real(srp), parameter, public food_item_density = 0.1_SRP

  The (physical) density of a single food item. TODO: need to parametrise!
- real(srp), parameter, public food_item_capture_prop_cost = 0.05_SRP

  The cost of the food item catching, in terms of the ***food item mass*** *(proportional cost). So, if the agent does an unsuccessful attempt to catch a food item, the cost still applies.*
- real(srp), parameter, public food_item_capture_probability = 0.99_SRP

  The ***baseline*** *probability that the food item is captured. See the_neurobio::food_item_capture_probability_calc().*
- real(srp), parameter, public food_item_capture_probability_min = 0.1_SRP

  The ***minimum*** *probability of capture a food item, when the item is at a distance equal to the visual range from the predator agent.*
- real(srp), parameter, public food_item_capture_probability_subjective_errorr_cv = 0.1

  Subjective error assessing the food item capture probability when assessing the subjective GOS expectancies of food items. The subjective assessment value of the capture probability is equal to the objective value plus random error with the CV equal to this parameter.
- real(srp), parameter, public food_item_migrate_xy_mean = FOOD_ITEM_SIZE_DEFAULT ∗ 10.0_SRP

  Mean shift parameter for the local random walk movement of food items in the horizontal plane.

- real(srp), parameter, public food_item_migrate_depth_mean = FOOD_ITEM_SIZE_DEFAULT * 100.0_SRP

  Mean shift parameter for the local random walk movement of food items in the vertical (depth) plane.
- real(srp), parameter, public food_item_migrate_xy_cv = FOOD_ITEM_SIZE_DEFAULT_CV

  Coefficient of variation parameter for the local random walk movement of food items in the horizontal plane.
- real(srp), parameter, public food_item_migrate_depth_cv = 0.8_SRP

  Coefficient of variation parameter for the local random walk movement of food items in the vertical (depth) plane.
- real(srp), parameter, public daylight =500.0_SRP

  Maximum above-surface light intensity at midday, DAYLIGHT=500.0.
- logical, parameter, public daylight_stochastic = .TRUE.

  Flag for stochastic daylight pattern (if TRUE) or deterministic sinusoidal (when FALSE). Check out the next parameter `DAYLIGHT_CV` for variability.
- real(srp), parameter, public daylight_cv =0.2_SRP

  Coefficient of variation for stochastic DAYLIGHT,.
- real(srp), parameter, public beamatt =1.0_SRP

  Beam attenuation coefficient of water (m-1),BEAMATT = 1.0.
- real(srp), parameter, public preycontrast_default = 1.0_SRP

  Inherent contrast of prey, CONTRAST =1.0.
- real(srp), parameter, public preyarea_default =3.E-6_SRP

  Area of prey (m2), PREYAREA = 3.E-6.
- real(srp), parameter, public viscap =1.6E6_SRP

  Dimensionless descriptor of fish eye quality, VISCAP=1.6E6.
- real(srp), parameter, public eyesat =500.0_SRP

  Saturation parameter of eye (Ke) (uE m-2 s-1), EYESAT=500.0.
- real(srp), parameter, public lightdecay = 0.002_SRP

  Vertical conservation of light, per depth (old code lightdecay=0.2).

Genetic architecture parameters

- integer, parameter, public allelerange_min = 1

  The minimum possible value of alleles (allele range minimum) See implementation notes on `the_genome::gene::allele_val` component of the `the_genome::gene` derived type and `commondata::alleleconv()` and `commondata::allelescale()` functions.
- integer, parameter, public allelerange_max = 10000

  The maximum possible value of alleles (allele range maximum) See implementation notes on `the_genome::gene::allele_va` component of the `the_genome::gene` derived type and `commondata::alleleconv()` and `commondata::allelescale()` functions.
- real(srp), parameter, public allelescale_max = 20.0_SRP

  Conversion parameter that defines the scaling of the integer allele values ::ALLELERANGE_MIN to ALLELERANGE_MAX are converted to zero to this parameter value as the maximum. See allelescale() for details.
- integer, parameter, public additive_comps = 3

  Number of additive allele components.
- real(srp), parameter, public mutationrate_point = 0.1_SRP

  Mutation rate for point allele mutations.
- real(srp), parameter, public ga_mutationrate_point_max = 0.25_SRP

  Maximum point mutation rate in the adaptive Fixed Fitness Genetic Algorithm.
- real(srp), parameter, public mutationrate_batch = 0.05_SRP

  Mutation rate for point allele mutations, a whole batch of allele components.
- real(srp), parameter, public ga_mutationrate_batch_max = 0.1_SRP

  Maximum batch mutation rate in the adaptive Fixed Fitness Genetic Algorithm.
- real(srp), parameter, public relocation_swap_rate = 0.05_SRP

  Mutation rate for chromosome relocation, i.e. probability of a gene moving to a different position on the same chromosome: There are two kinds of relocations, swapping genes between two positions and moving a gene with subsequent shift. So we have two constants for the respective rates.
- real(srp), parameter, public relocation_shift_rate = 0.01_SRP
- integer, parameter, public n_chromosomes = 6

  The number of chromosomes for the agents.
- integer, dimension(n_chromosomes), parameter, public len_chromosomes = [6, 5, 12, 12, 12, 12]

The number of alleles in each of the chromosomes. NOTE: This must be an array (vector) of the size `commondata::n_chromosomes`. We use new Fortran array constructor here to set the array values.

- integer, parameter, public max_nalleles = 12

  This parameter defines the maximum number of alleles within the chromosome It IS NOT intended to vary freely/independently. Used in definitions of `_GENOTYPE_PHENOTYPE` matrices, equal to the `maxval(LEN_CHROMOSOMES)`.
- character(len= ∗), dimension(n_chromosomes), parameter, public lab_chromosomes = ["C_1_SEX ", "C_2_BODY", "C_3_HORM", "C_4_HUNG", "C_5_FEAR", "C_6_REPR"]

  Set the labels of the chromosomes. NOTE, must be an array(vector)) of the size `commondata::n_chromosomes`. We use new Fortran array constructor here to set the array values.
- integer, parameter, public chromosome_ploidy = 2

  The ploidy of the chromosome set. Can theoretically be haploid (=1), diploid (=2) or, polyploid (>2).
- real(srp), parameter, public genome_recombination_ratio_mother = 0.8_SRP

  *The ratio of the genome that inherited from the **mother**. The other part is inherited from the father. See implementation details of the **random independent** genetic recombination procedure `procedures the_genome::individual_genome::recombine()`.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public genome_crossover_fixed_mother = reshape ([YES,YES,YES,YES,YES,YES, YES,YES,YES,YES,YES,YES, YES,YES,YES,YES,YES,YES, NO, NO,YES,YES,YES,YES, NO, NO,YES,YES,YES,YES, NO, NO,YES,YES,YES,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *Boolean 2D matrix that determines the pattern of **fixed** chromosome **crossover**. For each chromosome, the alleles that are marked with the TRUE (YES) values are inherited from the **mother** whereas those marked FALSE (NO) are inherited from the **father**. See implementation details of the **fixed genetic crossover** procedure `procedures the_genome::individual_genome::crossover()`.*
- real(srp), parameter, public sex_ratio = 0.5_SRP

  Sex ratio for initialising genomes.
- character(len=label_length), parameter sexlocus_label ="SEX_LOCUS"

  Labels for the sex locus alleles (gene) - vector as we don't need to label individual alleles. `LABEL_LENGTH` is here to avoid a GF warning.
- character(len= ∗), parameter, public male ="male"

  Set names of the sexes – the allele labels.
- character(len= ∗), parameter, public female ="female"
- logical, dimension(max_nalleles, n_chromosomes), parameter, public sex_genotype_phenotype = reshape ([YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *Sex definition can be implemented differently from all other traits. Here is an example of the **phenotype** x **genotype matrix** code for sex determination:*

Hormonal parameters

- logical, dimension(max_nalleles, n_chromosomes), parameter, public growhorm_genotype_phenotype = reshape ([NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *Genotype x Phenotype matrix for **growth hormone**.*
- real(srp), parameter, public growhorm_init = 0.5_SRP

  *Genotype to phenotype gamma2gene initialisation value for **growth hormone***
- real(srp), parameter, public growhorm_gerror_cv = 0.5_SRP

  Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).
- logical, dimension(max_nalleles, n_chromosomes), parameter, public thyroid_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *Genotype x Phenotype matrix for **thyroid**.*

- real(srp), parameter, public thyroid_init = 0.5_SRP

  *Genotype to phenotype gamma2gene initialisation value for **thyroid***
- real(srp), parameter, public thyroid_gerror_cv = 0.5_SRP

  Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).
- logical, dimension(max_nalleles, n_chromosomes), parameter, public adrenaline_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *Genotype x Phenotype matrix for **adrenaline***
- real(srp), parameter, public adrenaline_init = 0.5_SRP

  *Genotype to phenotype gamma2gene initialisation value for **adrenaline***
- real(srp), parameter, public adrenaline_gerror_cv = 0.5_SRP

  Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).
- logical, dimension(max_nalleles, n_chromosomes), parameter, public cortisol_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *Genotype x Phenotype matrix for **cortisol**.*
- real(srp), parameter, public cortisol_init = 0.5_SRP

  *Genotype to phenotype gamma2gene initialisation value for **cortisol***
- real(srp), parameter, public cortisol_gerror_cv = 0.5_SRP

  Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).
- logical, dimension(max_nalleles, n_chromosomes), parameter, public testosterone_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *Genotype x Phenotype matrix for **testosterone**.*
- real(srp), parameter, public testosterone_init = 0.01_SRP

  *Genotype to phenotype gamma2gene initialisation value for **testosterone***
- real(srp), parameter, public testosterone_gerror_cv = 0.5_SRP

  Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).
- logical, dimension(max_nalleles, n_chromosomes), parameter, public estrogen_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  Genotype x Phenotype matrix for ESTROGEN.
- real(srp), parameter, public estrogen_init = 0.01_SRP

  *Genotype to phenotype gamma2gene initialisation value for **estrogen***
- real(srp), parameter, public estrogen_gerror_cv = 0.5_SRP

  Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).
- integer, parameter, public sex_steroids_check_history = 3

  *The number of the latest historical values that are checked for change when setting an increment of the **sex steroid** hormones.*
- real(srp), dimension(∗), parameter, public sex_steroids_increment_factor_age_curve_abscissa = [0.0_↩ SRP, real(LIFESPAN, SRP)∗0.25_SRP, real(LIFESPAN, SRP)]

  *The array defining the **abscissa** (X) of the nonparametric function curve that defines the relationship between the age of the agent and the steroid increment factor for this specific age.*
- real(srp), dimension(∗), parameter, public sex_steroids_increment_factor_age_curve_ordinate = [0.0_↩ SRP, 0.01_SRP, 0.1_SRP]

  *The array defining the **ordinate** (Y) of the nonparametric function curve that defines the relationship between the age of the agent and the steroid increment factor for this specific age.*

- real(srp), dimension(∗), parameter, public sex_steroids_increment_factor_len_curve_abscissa = [0.0_↩ SRP, BODY_LENGTH_MAX∗0.2_SRP, BODY_LENGTH_MAX]

  *The array defining the **abscissa** (X) of the nonparametric function curve that defines the relationship between the body length of the agent and the steroid increment factor for this specific length.*
- real(srp), dimension(∗), parameter, public sex_steroids_increment_factor_len_curve_ordinate = [0.0_↩ SRP, 0.01_SRP, 0.1_SRP]

  *The array defining the **ordinate** (Y) of the nonparametric function curve that defines the relationship between the body length of the agent and the steroid increment factor for this specific length.*

Body parameters, individual physiology and condition

- integer, parameter, public history_size_agent_prop = 100

  History stack size for the agent's basic properties, such as body length and body mass. Normally they are saved only for the analysis and currently not used in the perception.
- real(srp), parameter, public living_cost = 4.0_SRP

  Living cost in terms of food consumed. metabolic costs, p roportional to body size.
- real(srp), parameter, public mass_growth_threshold = 0.0001_SRP

  A minimum body mass increment when any linear growth is possible, in units of the body mass (e.g. 0.05 = 5%)
- real(srp), parameter, public linear_growth_exponent = 3.0_SRP

  Growth exponent linking linear growth and body mass growth. Based on Fulton's condition factor "cube law.".
- real(srp), dimension(∗), parameter, public linear_growth_hormone_increment_factor_curve_abscissa = [0.0_SRP, GROWHORM_INIT, GROWHORM_INIT∗3.0_SRP, GROWHORM_INIT∗5.0_SRP, GROWHORM_INIT∗20.0_SRP]

  *The array defining the **abscissa** (X) of the nonparametric function curve that defines the function linking the relationship between the growth hormone and the relative linear growth increment.*
- real(srp), dimension(∗), parameter, public linear_growth_hormone_increment_factor_curve_ordinate = [0.0_SRP, 0.6_SRP, 0.9_SRP, 0.98, 1.00_SRP]

  *The array defining the **ordinate** (Y) of the nonparametric function curve that defines the function linking the relationship between the growth hormone and the relative linear growth increment.*
- real(srp), parameter, public max_stomach_capacity_def = 0.15_SRP

  Set the maximum stomach capacity default value – fraction of the body mass available for food. Can be overriden in different agent types. Normally 15%.
- real(srp), parameter, public stomach_content_emptify_factor = 100.0_SRP

  Stomach content emptify factor at each step.
- real(srp), parameter, public stomach_content_init = 0.01_SRP

  Set average stomach capacity at birth/init in units of body weight,.
- real(srp), parameter, public stomach_content_init_cv = 0.05_SRP

  Set the coefficient of variation for the stomach capacity at init.
- real(srp), parameter, public swimming_speed_cost_burst = 0.1E-4_SRP

  Set the weighting factor parameter of burst swimming cost in terms of the agent body size and the distance expressed in terms of the agent body lengths. In the_body::condition_cost_swimming_burst(), this parameter sets the β coefficient.
- real(srp), parameter, public cost_factor_foraging_smr = 0.1E-5_SRP

  Set the cost of foraging in terms of SMR.
- logical, dimension(max_nalleles, n_chromosomes), parameter, public energy_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The initial value of the **energy reserves** at birth is genetically determined. This is the Genotype x Phenotype matrix for energy reserves.*
- real(srp), parameter, public energy_init = 0.18

  This is the initial value of the energy reserves, non-genetic mean.
- real(srp), parameter, public energy_gerror_cv = 0.01

  Genotype to phenotype initialisation, Gaussian error parameter. Coefficient of variation for the `ENERGY_INIT` value.
- logical, dimension(max_nalleles, n_chromosomes), parameter, public body_length_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

The initial value of the body length at birth is genetically determined. This is the Genotype x Phenotype matrix for body length.

- real(srp), parameter, public body_length_init = 2.3_SRP

  *The initial value of **body length**, the average (gon-genetic).*
- real(srp), parameter, public body_length_gerror_cv = 0.1_SRP

  Genotype to phenotype initialisation, Gaussian error parameter. Coefficient of variation for the `BODY_LENGTH_INIT` *value.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public control_unselected_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The initial value of the **control unselected** trait. This trait is is genetically determined but is not selected or used. So it can be used to control for random genetic drift. This is the Genotype x Phenotype matrix.*
- real(srp), parameter, public control_unselected_init = 0.5_SRP

  *The initial value of the **control unselected** trait that goes through the gamma2gene.*
- real(srp), parameter, public control_unselected_gerror_cv = 0.5_SRP

  Genotype to phenotype initialisation, Gaussian error parameter. Coefficient of variation for the control unselected trait.
- logical, dimension(max_nalleles, n_chromosomes), parameter, public smr_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  The initial value of the standard metabolic rate (SMR) at birth is genetically determined. This is the Genotype x Phenotype matrix for SMR.
- real(srp), parameter, public smr_init = 0.5_SRP

  This is the initial value of SMR that goes through the gamma2gene.
- real(srp), parameter, public smr_gerror_cv = 0.5_SRP

  Genotype to phenotype initialisation, Gaussian error parameter. Coefficient of variation for the `SMR_LENGTH_INIT` *value.*
- real(srp), parameter, public smr_min = 0.01_SRP

  Minimum SMR value, anything lower is not allowed.
- real(srp), parameter swimming_cost_exponent_laminar = 0.5_SRP

  *Default swimming cost body mass exponent parameter for **laminar** flow. See doi:10.1242/jeb.01484 (* `https://dx.doi.org/10.1242/jeb.01484`*) and the_body::condition_cost_swimming_burst() for details.*
- real(srp), parameter swimming_cost_exponent_turbulent = 0.6_SRP

  *Default swimming cost body mass exponent parameter for **turbulent** flow. See doi:10.1242/jeb.01484 (* `https://dx.doi.org/10.1242/jeb.01484`*) and the_body::condition_cost_swimming_burst() for details.*
- real(srp), parameter, public swimming_cost_factor_buoyancy_down = 0.01_SRP

  This parameter defines the cost of the buoyancy-based locomotion as a fraction of normal laminar flow propulsion for lowering downwards.
- real(srp), parameter, public swimming_cost_factor_buoyancy_up = 0.1_SRP

  This parameter defines the cost of the buoyancy-based locomotion as a fraction of normal laminar flow propulsion for lowering downwards.

Parameters of the neurobiological architecture

- integer, parameter, public food_select_items_index_partial = 20

  *Sets the limit for partial indexing and ranking of **food items** in the visual range of the agents.*
- integer, parameter, public consp_select_items_index_partial = 20

  *Sets the limit for partial indexing and ranking of **conspecifics** in the visual range of the agent.*
- integer, parameter, public pred_select_items_index_partial = 20

  *Sets the limit for partial indexing and ranking of **predators** in the visual range of the agent.*
- real(srp), parameter, public individual_visual_contrast_default = 1.0_SRP

  *Inherent contrast of the **agent**, It is used in determining the visual range of an agent in perception of conspecifics, and also for assessing the agent's detectability by predator. Default Contrast of food items is set separately.*
- integer, parameter, public history_size_perception = HISTORY_SIZE_SPATIAL

  Sets the size of the perception memory stack.
- integer, parameter, public history_size_motivation = HISTORY_SIZE_SPATIAL

Sets the size of the emotional state memory stack.

- logical, dimension(max_nalleles, n_chromosomes), parameter, public light_hunger_genotype_neuronal = reshape ([NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

    *The genotype structure for **light** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public light_hunger_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

    *Gaussian perception error parameter (cv) for **light** perception effects on **hunger**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public depth_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

    *The genotype structure for **depth** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public depth_hunger_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

    *Gaussian perception error parameter (cv) for **depth** perception effects on **hunger**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public foodcount_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

    *The genotype structure for **food items count** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public foodcount_hunger_genotype_neuronal_gerror_cv = PERCEPT_ERROR_↩ CV_DEF

    *Gaussian perception error parameter (cv) for **food items count** perception effects on **hunger**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public food_mem_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

    *The genotype structure for **food items count** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public food_mem_hunger_genotype_neuronal_gerror_cv = PERCEPT_ERROR_↩ CV_DEF

    *Gaussian perception error parameter (cv) for **food items count** perception effects on **hunger**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public conspcount_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

    *The genotype structure for **conspecifics number** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public conspcount_hunger_genotype_neuronal_gerror_cv = PERCEPT_ERROR_↩ CV_DEF

    *Gaussian perception error parameter (cv) for **conspecifics number count** perception effects on **hunger**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public pred_direct_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

    *The genotype structure for **direct predation** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public pred_direct_hunger_genotype_neuronal_gerror_cv = PERCEPT_ERROR_↩ CV_DEF

    *Gaussian perception error parameter (cv) for **direct predation** perception effects on **hunger**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public pred_meancount_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **mean predator number** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public pred_meancount_hunger_genotype_neuronal_gerror_cv = PERCEPT_↩ ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **mean predator number** perception effects on **hunger**.*
- real(srp), parameter, public predation_risk_weight_immediate = 0.7

  The weight of the immediately seen predators over those in the perceptual memory stack. Those in the memory then have this weight in the predation risk estimate: `1-PREDATION_RISK_WEIGHT_IMMEDIATE`. *See the_neurobio::predation_risk_backend().*
- integer, parameter, public predation_risk_weight_memory_window = nint(HISTORY_SIZE_PERCEPTION / 2.0)

  The length of the perceptual memory window that is taken into account and weighted against the currently seen number of predators when calculating the motivation value.
- logical, dimension(max_nalleles, n_chromosomes), parameter, public stom_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **stomach** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public stom_hunger_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **stomach** perception effects on **hunger**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public bodymass_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **body mass** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public bodymass_hunger_genotype_neuronal_gerror_cv = PERCEPT_ERROR_↩ CV_DEF

  *Gaussian perception error parameter (cv) for **body mass** perception effects on **hunger**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public energy_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **energy reserves** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public energy_hunger_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_↩ DEF

  *Gaussian perception error parameter (cv) for **energy reserves** perception effects on **hunger**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public age_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **age** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public age_hunger_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **age** perception effects on **hunger**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public reprfac_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

*The genotype structure for **reproductive factor** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public reprfac_hunger_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_↩
DEF

  *Gaussian perception error parameter (cv) for **reproductive factor** perception effects on **hunger**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public light_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **light** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public light_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV↩
_DEF

  *Gaussian perception error parameter (cv) for **light** perception effects on **fear state**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public depth_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **depth** perception effects on **active avoidance** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public depth_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_↩
CV_DEF

  *Gaussian perception error parameter (cv) for **depth** perception effects on **fear state**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public foodcount_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **food items count** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public foodcount_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR↩
_CV_DEF

  *Gaussian perception error parameter (cv) for **food items count** perception effects on **fear state**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public food_mem_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **food items count** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public food_mem_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_↩
ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **food items count** perception effects on **fear state**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public conspcount_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **conspecifics number** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public conspcount_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_↩
ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **conspecifics number count** perception effects on **fear state**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public pred_direct_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

*The genotype structure for **direct predation** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public pred_direct_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_↩ ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **direct predation** perception effects on **fear state**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public pred_meancount_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **mean predator number** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public pred_meancount_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_↩ ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **mean predator number** perception effects on **fear state**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public stom_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **stomach** perception effects on **active avoidance** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public stom_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV↩ _DEF

  *Gaussian perception error parameter (cv) for **stomach** perception effects on **fear state**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public bodymass_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **body mass** perception effects on **active avoidance** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public bodymass_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_↩ ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **body mass** perception effects on **fear state**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public energy_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **energy reserves** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public energy_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_↩ CV_DEF

  *Gaussian perception error parameter (cv) for **energy reserves** perception effects on **fear state**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public age_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **age** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public age_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_↩ DEF

  *Gaussian perception error parameter (cv) for **age** perception effects on **fear state**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public reprfac_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

*The genotype structure for **reproductive factor** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public reprfac_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_↩ CV_DEF

  *Gaussian perception error parameter (cv) for **reproductive factor** perception effects on **fear state**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public light_reproduce_genotype_neuronal = reshape ([NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **light** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public light_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_↩ DEF

  *Gaussian perception error parameter (cv) for **light** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public depth_reproduce_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **depth** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public depth_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV↩ _DEF

  *Gaussian perception error parameter (cv) for **depth** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public foodcount_reproduce_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **food items count** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public foodcount_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR↩ _CV_DEF

  *Gaussian perception error parameter (cv) for **food items count** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public food_mem_reproduce_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **food items count** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public food_mem_reproduce_genotype_neuronal_gerror_cv = PERCEPT_↩ ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **food items count** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public conspcount_reproduce_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **conspecifics number** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public conspcount_reproduce_genotype_neuronal_gerror_cv = PERCEPT_↩ ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **conspecifics number count** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public pred_direct_reproduce_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

*The genotype structure for **direct predation** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public pred_direct_reproduce_genotype_neuronal_gerror_cv = PERCEPT_↩ ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **direct predation** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public pred_meancount_reproduce_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **mean predator number** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public pred_meancount_reproduce_genotype_neuronal_gerror_cv = PERCEPT_↩ ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **mean predator number** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public stom_reproduce_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **stomach** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public stom_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV↩ _DEF

  *Gaussian perception error parameter (cv) for **stomach** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public bodymass_reproduce_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **body mass** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public bodymass_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR↩ _CV_DEF

  *Gaussian perception error parameter (cv) for **body mass** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public energy_reproduce_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **energy reserves** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public energy_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR_↩ CV_DEF

  *Gaussian perception error parameter (cv) for **energy reserves** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public age_reproduce_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **age** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public age_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_↩ DEF

  *Gaussian perception error parameter (cv) for **age** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public reprfac_reproduce_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

*The genotype structure for **age** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public reprfac_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR_↩ CV_DEF

  *Gaussian perception error parameter (cv) for **age** perception effects on **reproduction**.*
- logical, parameter, public modulation_appraisal_disable_all = .FALSE.

  *Disable developmental or genetic modulation of the `APPRAISAL` completely. If set to TRUE, **no modulation** is performed and the final motivation values are just equal to the primary modulation values in `APPRAISAL`.*
- real(srp), parameter, public reprod_modulation_devel_agestart = LIFESPAN / 2.0_SRP

  Developmental modulation of reproductive motivation. This parameter defines the starting age (time steps) at which reproductive motivation can rise above zero (at all previous ages weight=0.0). See the_neurobio::appraisal_motivation_modulation_non_genetic().
- real(srp), parameter, public reprod_modulation_devel_agefull = REPROD_MODULATION_DEVEL_↩ AGESTART + REPROD_MODULATION_DEVEL_AGESTART ∗ 2.0_SRP / 3.0_SRP

  Developmental modulation of reproductive motivation. This parameter defines the age (time steps) at which reproductive motivation is weighted fully (weight = 1.0). See the_neurobio::appraisal_motivation_modulation_non_genetic().
- real(srp), dimension(∗), parameter reprod_modulation_devel_abscissa = [REPROD_MODULATION↩ _DEVEL_AGESTART, REPROD_MODULATION_DEVEL_AGESTART + (REPROD_MODULATION↩ _DEVEL_AGEFULL - REPROD_MODULATION_DEVEL_AGESTART) / 3.0_SRP, REPROD_↩ MODULATION_DEVEL_AGEFULL]

  Developmental modulation of reproductive motivation. This is the interpolation grid abscissa defining at which age reproduction motivation can have non-zero values (reproduction is possible). It defines the weighting factor applied to reproductive motivation.
- real(srp), parameter, public reprod_modulation_devel_w2 = 0.1_SRP

  Developmental modulation of reproductive motivation. This parameter sets the interpolation array weight that defines how fast the reproduction motivation the_neurobio::state_reproduce is allowed to raise when the age of the agent exceeds the reproductive age. For details see the_neurobio::appraisal_motivation_modulation_non_genetic().
- logical, dimension(max_nalleles, n_chromosomes), parameter, public sex_male_modulation_reproduce_genotype = reshape ([NO, NO, NO, NO, NO, NO, YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for sex modulation coefficient affecting **reproduction** motivation state in **males**.*
- real(srp), parameter, public sex_male_modulation_reproduce_gerror_cv = 0.1_SRP

  *Gaussian error parameter (cv) for the sex modulation coefficient affecting **reproduction** motivation state in **males**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public sex_female_modulation_reproduce_genotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for sex modulation coefficient affecting **reproduction** motivation state in **females**.*
- real(srp), parameter, public sex_female_modulation_reproduce_gerror_cv = 0.1_SRP

  *Gaussian error parameter (cv) for the sex modulation coefficient affecting **reproduction** motivation state in **females**.*
- real(srp), parameter, public attention_switch_hunger_light = 0.0_SRP

  Baseline attention switches $\Psi_{i,j}$ control which perception components (i) can affect each of the motivational state (j). They should be defined for all combinations between the perception components (light, depth... food... predators... etc) and the motivational states. See Cognitive architecture.
- real(srp), parameter, public attention_switch_hunger_depth = 0.0_SRP

  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_hunger_food_dir = 1.0_SRP

  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_hunger_food_mem = 1.0_SRP

  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_hunger_conspec = 1.0_SRP

  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_hunger_pred_dir = 0.0_SRP

  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_hunger_predator = 0.0_SRP

  Baseline attention switch, see commondata::attention_switch_hunger_light.

- real(srp), parameter, public attention_switch_hunger_stomach = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_hunger_bodymass = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_hunger_energy = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_hunger_age = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_hunger_reprfac = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_avoid_act_light = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_avoid_act_depth = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_avoid_act_food_dir = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_avoid_act_food_mem = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_avoid_act_conspec = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_avoid_act_pred_dir = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_avoid_act_predator = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_avoid_act_stomach = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_avoid_act_bodymass = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_avoid_act_energy = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_avoid_act_age = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_avoid_act_reprfac = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_reproduce_light = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_reproduce_depth = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_reproduce_food_dir = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_reproduce_food_mem = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_reproduce_conspec = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_reproduce_pred_dir = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_reproduce_predator = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_reproduce_stomach = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_reproduce_bodymass = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_reproduce_energy = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_reproduce_age = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public attention_switch_reproduce_reprfac = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.

- real(srp), dimension(*), parameter, public attention_modulation_curve_abscissa =[0.0_SRP, 0.3_SRP, 0.5_SRP, 1.0_SRP]

  *The array defining the **abscissa** (X) of the nonparametric function that defines the **attention modulation curve** by the current Global Organismic State (GOS).*
- real(srp), dimension(*), parameter, public attention_modulation_curve_ordinate =[1.0_SRP, 0.98_SRP, 0.9_SRP, 0.0_SRP]

  *The array defining the **ordinate** (Y) of the nonparametric function that defines the **attention modulation curve** by the current Global Organismic State (GOS).*
- real(srp), dimension(*), parameter, public motivation_compet_threshold_curve_abscissa = [0.0_SRP, 0.↩ 2_SRP, 0.60_SRP, 0.80_SRP, 0.90_SRP, 1.0_SRP, 1.1_SRP]

  *The array defining the **abscissa** (X) of the nonparametric function curve that defines the threshold for motivation competition in GOS.*
- real(srp), dimension(*), parameter, public motivation_compet_threshold_curve_ordinate = [1.0_SRP, 0.↩ 3_SRP, 0.04_SRP, 0.01_SRP, 0.001_SRP, 0.0_SRP, 0.0_SRP]

  *The array defining the **ordinate** (Y) of the nonparametric function curve that defines the threshold for motivation competition in GOS.*
- real(srp), parameter, public arousal_gos_dissipation_factor = 0.5_SRP

  *Spontaneous arousal dissipation level when a simple **fixed** dissipation factor pattern is used. At each step, `gos↩ _arousal` is reduced by a constant factor, AROUSAL_GOS_DISSIPATION_FACTOR` (e.g. reduced by 0.5) independently on the current GOS time step.*
- real(srp), dimension(*), parameter, public arousal_gos_dissipation_nonpar_abscissa = [1.0, 2.00, 5.00, 10.0, 15.0, 18.0, 20.0]

  *This is the array defining the **abscissa** (X) of the nonparametric spontaneous arousal dissipation factor function involving polynomial (or linear) interpolation is used.*
- real(srp), dimension(*), parameter, public arousal_gos_dissipation_nonpar_ordinate = [1.0, 0.98, 0.80, 0.40, 0.22, 0.18, 0.17]

  *This is the array defining the **ordinate** (Y) of the nonparametric spontaneous arousal dissipation factor function involving polynomial (or linear) interpolation is used.*
- real(srp), public global_rescale_maximum_motivation

  Global maximum sensory information that is updated for the whole population of agents.
- integer, parameter, public history_size_behaviours = HISTORY_SIZE_SPATIAL

  The size of the behaviour labels history stack, i.e. for how many time steps should the stack remember record the behaviour labels.
- real(srp), parameter, public probability_reproduction_base_factor = 0.90

  Default weighting factor for the baseline probability of successful reproduction φ. See implementation details for the function the_neurobio::reproduce_do_probability_reproduction_calc().
- real(srp), dimension(*), parameter, public probability_reproduction_delta_mass_abscissa = [0.5_SRP, 1.↩ 0_SRP, 2.0_SRP]

  *Interpolation grid **abscissa** for the body mass ratio factor that scales the probability of reproduction. For details see the_neurobio::reproduce_do_probability_reproduction_calc() procedure. Commands (template) to produce interpolation plots:*
- real(srp), dimension(*), parameter, public probability_reproduction_delta_mass_ordinate = [0.0_SRP, 1.↩ 0_SRP, 1.8_SRP]

  *Interpolation grid **ordinate** for the body mass ratio factor that scales the probability of reproduction. For details see the_neurobio::reproduce_do_probability_reproduction_calc() procedure. Commands (template) to produce interpolation plots:*
- real(srp), parameter, public sex_steroids_reproduction_threshold = 1.3_SRP

  This parameter defines the threshold of the current gonadal steroids level that should exceed the baseline value determined by the genome, for reproduction to be possible.
- real(srp), parameter, public walk_random_distance_default_factor = 10.0_SRP

  The weighting factor used in calculation of the default random walk distance, in terms of the agent's body length.
- real(srp), parameter, public walk_random_distance_stochastic_cv = 0.5_SRP

  The coefficient of variation of the distance for stochastic Gaussian random walk (distance is in terms of the agent's body length). Note that for deterministic walk, cv is zero.
- real(srp), parameter, public walk_random_food_gain_hope = 4.0_SRP

  *The maximum walk distance, **in units of the average distance to food items** in the current perception object, when the expected food gain is calculated on the bases of the current food availability, not using the the_behaviour::hope() function mechanism. If the average walk distance exceeds this value, the expectancy is based on the the_behaviour::hope() function.*
- real(srp), parameter, public walk_random_food_gain_hope_agentl = 100.0_SRP

*The maximum walk distance, **in units of the agent body length**, when the expected food gain is calculated on the bases of the current food availability, not using the the_behaviour::hope() function mechanism. If the average walk distance exceeds this value, the expectancy is based on the the_behaviour::hope() function.*

- real(srp), parameter, public walk_random_pred_risk_hope_agentl = 150.0_SRP

  *The maximum walk distance, **in units of the agent body length**, when the expected predation risk is calculated on the basis of the current perception value, not using the the_behaviour::hope() function mechanism. If the average walk distance exceeds this value, the risk expectancy is based on the the_behaviour::hope() function.*

- real(srp), parameter, public walk_random_vertical_shift_ratio = 0.5_SRP

  The ratio of the vertical to main horizontal shift parameters of the agent's Gaussian random walk. Random walk is done in the "2.5D" mode (the_environment::spatial_moving::rwalk25d()), i.e. with separate parameters for the main horizontal shift and the vertical depth shift. This is done to avoid a potentially too large vertical displacement of the agent during the movement. Thus, the vertical shift distance should normally be smaller than the horizontal shift. The difference between the main horizontal and (the smaller) vertical shifts is defined by this parameter. For example, if it is equal to 0.5, then the vertical depth shift is 0.5 of the main horizontal shift. See the_behaviour::walk_random_do_execute() for more details.

- real(srp), parameter, public walk_random_vertical_shift_cv_ratio = 1.0_SRP

  *The ratio of the vertical to the main horizontal coefficients of variation for the **vertical** depth distance in the stochastic Gaussian random walk of the agent. Should normally be equal to the main default value set by commondata::walk_random_distance_stochastic_cv. That is 1.0.*

- real(srp), dimension(*), parameter, public walk_random_food_hope_abscissa = [0.0_SRP, 1.0_SRP, 3.↩ 5_SRP]

  This parameter defines the hope function for calculating the food perception expectancy in the the_behaviour::walk_random behaviour. This is the abscissa for the hope function grid array. Plotting: `htintrpl.exe [0 1 3.5 ] [2, 1, 0]`. See the_behaviour::walk_random_do_this().

- real(srp), dimension(*), parameter, public walk_random_food_hope_ordinate = [2.0_SRP, 1.0_SRP, 0.↩ 0_SRP]

  This parameter defines the hope function for calculating the food perception expectancy in the the_behaviour::walk_random behaviour. This is the ordinate for the hope function grid array. Plotting: `htintrpl.exe [0 1 3.5 ] [2, 1, 0]`. See the_behaviour::walk_random_do_this().

- real(srp), parameter, public approach_offset_default = TOLERANCE_HIGH_DEF_SRP

  Default offset for approach, offset is the difference between the approaching agent and the target object.

- real(srp), parameter, public approach_conspecfic_dilute_general_risk = 0.5_SRP

  Multiplication factor for the general risk of predation used when the agent evaluates the approach to a target conspecific.

- real(srp), parameter, public approach_conspecfic_dilute_adjust_pair_behind = 0.5_SRP

  Multiplication factor for subjective assessment of the direct risk of predation when the actor agent moves behind the target conspecific, i.e. when the distance between the agent and predator is going to become longer than the distance between the target conspecific and the agent. See the_behaviour::approach_conspecifics_do_this() for details.

- real(srp), dimension(*), parameter, public approach_food_gain_compet_factor_abscissa = [0.00_SRP, 0.10_SRP, 1.00_SRP, 1.50_SRP]

  *The grid **abscissa** defining the nonparametric relationship that determines the expected food gain for the "approach conspecifics" behaviour (the_behaviour::approach_conspec class). The function is a weighting factor depending on the ratio of the agent body mass to the target conspecific body mass, for the baseline expected food gain.*

- real(srp), dimension(*), parameter, public approach_food_gain_compet_factor_ordinate = [0.00_SRP, 0.01_SRP, 0.50_SRP, 1.00_SRP]

  *The grid **ordinate** defining the nonparametric relationship that determines the expected food gain for the "approach conspecifics" behaviour (the_behaviour::approach_conspec class). The function is a weighting factor depending on the ratio of the agent body mass to the target conspecific body mass, for the baseline expected food gain.*

- real(srp), parameter, public dist_expect_food_uncertain_fact = 0.7_SRP

  The weighting factor for the distance to the expected food item if the actual distance is uncertain (e.g. no food items currently in perception). See the_behaviour::walk_random_motivations_expect().

- real(srp), parameter, public history_perception_window_pred = 0.3_SRP

  The size of the memory window that is used in the assessment of predation risk, as a portion of the commondata::history_size_perception. See the_behaviour::walk_random_do_this() and the_behaviour::walk_random_motivations_ex

- real(srp), parameter, public history_perception_window_food = 0.3_SRP

  The size of the memory window that is used in the assessment of food gain, as a portion of the commondata::history_size_perception. See the_behaviour::walk_random_do_this() and the_behaviour::walk_random_motivations_ex

- real(srp), parameter, public escape_dart_distance_default_factor = 1.5_SRP

The weighting factor used in calculation of the default escape distance. The escape distance is equal to the visibility range of the predator multiplied by this factor. Therefore, it should normally exceed 1.0. Otherwise, the escaping object is still within the visibility range of the predator after the escape. See the_behaviour::escape_dart_do_this() for more details.

- real(srp), parameter, public escape_dart_distance_default_stoch_cv = 0.5_SRP

  For stochastic escape, this parameter determines the coefficient of variation of the escape walk. See the_behaviour::escape_dart_do_this() for more details.
- real(srp), parameter, public up_down_walk_step_stdlength_factor = 4.0_SRP

  The default size of the up and down walks performed by the GO_DOWN_DEPTH and GO_UP_DEPTH, see the_behaviour::go_down_depth and the_behaviour::go_up_depth classes as well as the_behaviour::go_down_do_this() and the_behaviour::go_up_do_this() methods.
- real(srp), parameter, public migrate_dist_max_step = 800.0_SRP

  The maximum distance (in units of the agent body length) a migrating agent can pass for a single time step of the model. This is basically limited by (an implicit) maximum speed of the agent, in terms of its body length. This parameter sets the limit on the length of a single migration bout.
- real(srp), parameter, public migrate_random_max_dist_target = 10.0_SRP

  Default maximum distance towards the target environment (in units of the agent's body size) when the agent could emigrate into this target environment. See the_behaviour::behaviour_do_migrate_random() for details.
- real(srp), parameter, public migrate_dist_penetrate_offset = 1.0_SRP

  The offset, in terms of the body length of the actor agent, for initial penetrating into the target environment when the agent is migrating into this environment. See the_environment::migrate_do_this().
- real(srp), parameter, public migrate_food_gain_maximum_hope = 2.0_SRP

  This parameter defines the hope function for calculating the food gain expectancy in the migration behaviour. This is the maximum value of the hope function that is achieved at zero ratio of the old to new food gain memory values. Plotting: `htintrpl.exe [0 1 3.5] [2 1 0]`. See the_behaviour::migrate_do_this().
- real(srp), parameter, public migrate_food_gain_ratio_zero_hope = 3.5_SRP

  This parameter defines the hope function for calculating the food gain expectancy in the migration behaviour. This is the maximum ratio of the old to new food gain memory values that leads to virtually zero value of the hope function. Plotting: `htintrpl.exe [0 1 3.5] [2 1 0]`. See the_behaviour::migrate_do_this().
- real(srp), parameter, public migrate_predator_maximum_hope = 2.0_SRP

  This parameter defines the hope function for calculating the general predation risk expectancy in the migration behaviour. This is the maximum value of the hope function that is achieved at zero ratio of the old to new predation values in the memory stack. Plotting: `htintrpl.exe [0 1 3.5] [2 1 0]`. See the_behaviour::migrate_do_this().
- real(srp), parameter, public migrate_predator_zero_hope = 3.5_SRP

  This parameter defines the hope function for calculating the general predation risk expectancy in the migration behaviour. This is the maximum ratio of the old to new predation values in the memory stack that leads to virtually zero value of the hope function. Plotting: `htintrpl.exe [0 1 3.5] [2 1 0]`. See the_behaviour::migrate_do_this().
- real(srp), dimension(∗), parameter, public behav_walk_step_stdlen_static = [1.0_SRP, 10.0_SRP, 25.0↩ _SRP, 50.0_SRP, 100.0_SRP]

  This parameter array defines the repertoire of predetermined static walk step sizes, in units of the agent's body length, for the the_behaviour::walk_random behavioural unit as executed in the the_behaviour::behaviour::walk_random class level. See the_behaviour::behaviour::select() method for details.
- real(srp), dimension(∗), parameter, public behav_go_up_down_step_stdlen_static = [10.0_SRP, 20.0_↩ SRP, 50.0_SRP, 75.0_SRP, 100.0_SRP]

  This parameter array defines the step sizes, in units of the agent's body length, for the the_behaviour::go_down_depth and the_behaviour::go_up_depth behavioural unit as executed in the the_behaviour::behaviour::depth_down and the_behaviour::behaviour::depth_up class level(s). See the_behaviour::behaviour::select() method for details.

Parameters of the Genetic Algorithm

- real(srp), parameter, public ga_reproduce_pr = 0.05_SRP

  Percentage of the best reproducing agents in the pre-evolution phase.
- integer, parameter, public ga_reproduce_n = int(POPSIZE ∗ GA_REPRODUCE_PR)

  Upper limit on the number of reproducing individuals in the fixed-fitness pre-evolution phase.
- integer, parameter, public ga_fitness_dead = 400000000

  Fitness value ascribed to dead agent in pre-evol. See the_individual::individual_agent::fitness_calc(). Also note that huge(integer)) = 2147483647.
- integer, parameter, public ga_fitness_select = 900

  Fitness threshold for the inclusion of the agent into the reproducing elite group.

- real(srp), parameter, public ga_reproduce_min_prop = 0.05_SRP

  Minimum proportion of reproducing agents, but note that the number of number reproducers cannot be smaller than the absolute minimum commondata::ga_reproduce_n_min. See the_population::population::ga_reproduce_max().
- integer, parameter, public ga_reproduce_n_min = 20

  Absolute minimum number of reproducing agents in the adaptive GA procedure. See the_population::population::ga_reproduce_max().

8.1.1 Detailed Description

COMMONDATA – definitions of global constants and procedures.

8.1.2 Commondata module

Module **COMMONDATA** is used for defining various global parameters like model name, tags, population size etc. Everything that has global scope and should be passed to many subroutines/functions, should be defined in `COMMONDATA`. It is also safe to include public keyword to declarations. `COMMONDATA` may also include subroutines/functions that have general scope and used by many other modules of the model.

8.1.2.0.1 Accessibility of objects By default, all **data objects** in `COMMONDATA` should be accessible to all other modules. However, procedures defined here must have granular access rights, with the generic name declared *public* while specific implementations *private*.

8.1.3 Function/Subroutine Documentation

8.1.3.1 cm2m_r()

```
elemental real(srp) function, private commondata::cm2m_r (
            real(srp), intent(in) value_cm )  [private]
```

Convert cm to m.

Parameters

value_cm	value in cm

Returns

value in m

Note

This version gets real type argument.

This is needed because some of the the sizes are expressed in cm but certain functions (e.g. visual range estimator SRGETR) require parameters in m. E.g. FOOD_ITEM_SIZE_DEFAULT is set around 0.5 cm while SRGETR requires prey area in m^2.
Definition at line 5550 of file m_common.f90.

8.1.3.2 cm2m_hr()

```
elemental real(hrp) function, private commondata::cm2m_hr (
            real(hrp), intent(in) value_cm )  [private]
```

Convert cm to m.

Parameters

value_cm	value in cm

Returns

value in m

Note

This version uses the **high** HRP numerical precision model.

Definition at line 5564 of file m_common.f90.

8.1.3.3 cm2m_i()

```
elemental real(srp) function, private commondata::cm2m_i (
            integer, intent(in) value_cm )  [private]
```

Convert cm to m.

Returns

value in m

Parameters

value_cm	value in cm

Note

This version gets integer argument (albeit returns real).

Definition at line 5578 of file m_common.f90.

8.1.3.4 m2cm_r()

```
elemental real(srp) function, private commondata::m2cm_r (
            real(srp), intent(in) value_m )  [private]
```

Convert m to cm.

Parameters

value_cm	value in cm

Returns

value in m

Note

This version gets real type argument.

Definition at line 5592 of file m_common.f90.

8.1.3.5 m2cm_hr()

```
elemental real(hrp) function, private commondata::m2cm_hr (
            real(hrp), intent(in) value_m )  [private]
```

Convert m to cm.

Returns

value in m

Parameters

value_cm	value in cm

Note

This version uses the **high** HRP numerical precision model.

Definition at line 5606 of file m_common.f90.

8.1.3.6 m2cm_i()

```
elemental real(srp) function, private commondata::m2cm_i (
            integer, intent(in) value_m )  [private]
```

Convert m to cm.

Parameters

value_cm	value in cm

Returns

value in m

Note

This version gets integer argument (albeit returns real).

Definition at line 5620 of file m_common.f90.

8.1.3.7 mm2m_r()

```
elemental real(srp) function, private commondata::mm2m_r (
            real(srp), intent(in) value_mm )  [private]
```

Convert mm to m.

Parameters

value_cm	value in cm

Returns

value in m

Note

This version gets real type argument.

Definition at line 5634 of file m_common.f90.

8.1.3.8 mm2m_i()

```
elemental real(srp) function, private commondata::mm2m_i (
            integer, intent(in) value_mm )  [private]
```

Convert mm to m.

Parameters

value_cm	value in cm

Returns

value in m

Note

This version gets integer argument (albeit returns real).

Definition at line 5648 of file m_common.f90.

8.1.3.9 carea()

```
elemental real(srp) function commondata::carea (
            real(srp), intent(in) R )
```

Calculate a circle area.
Calculate the area of the circle.

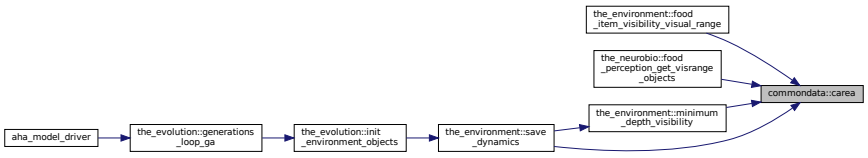
Parameters

R	circle radius.

Note

Food items are viewed as circular objects with the default **radius** equal to the **size** FOOD_ITEM_SIZE_↩ DEFAULT or FOOD_ITEM_MEAN_SIZE.

Definition at line 5664 of file m_common.f90.
Here is the caller graph for this function:



8.1.3.10 length2sidearea_fish()

```
elemental real(srp) function commondata::length2sidearea_fish (
            real(srp), intent(in) body_length )
```

A function linking **body length** with the body **area** in fish.
For fish, based on the paper by O'Shea et al., 2006, DOI: 10.1111/j.1365-2761.2006.00728.x Approximate formula for the the **whole surface area** is

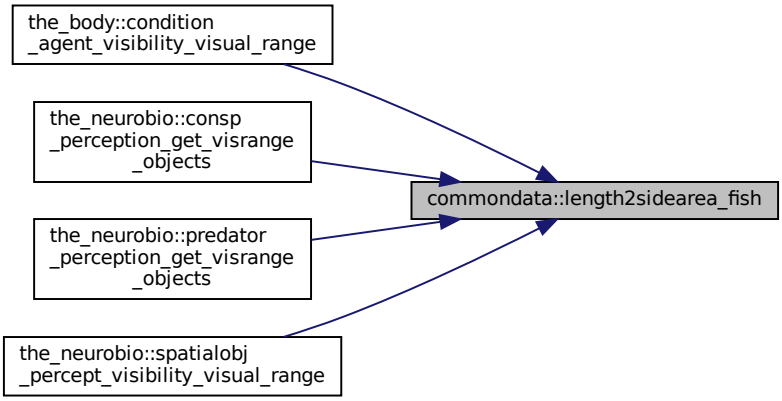
$$S = 0.7 \cdot L^{1.9}.$$

Because the side area is 1/2 of the total surface area (0.7 / 2 = 0.35), the function takes this half. So the final formula is:

$$S = 0.35 \cdot L^{1.8}.$$

Definition at line 5681 of file m_common.f90.

Here is the caller graph for this function:



8.1.3.11 rescale_full()

```
elemental real(srp) function, private commondata::rescale_full (
            real(srp), intent(in) value_in,
            real(srp), intent(in) A,
            real(srp), intent(in) B,
            real(srp), intent(in) A1,
            real(srp), intent(in) B1 )  [private]
```

Rescale a real variable with the range A:B to have the new range A1:B1.
Linear transformation of the input value `value_in` such `k * value_in + beta`, where the `k` and `beta` coefficients are found by solving a simple linear system: $\{ A_1 = k \cdot A + \beta; B_1 = k \cdot B + \beta$. It has this solution: $k = \frac{A_1 - B_1}{A - B}, \beta = -\frac{A_1 \cdot B - A \cdot B_1}{A - B}$

Warning

The function does not check if `value_in` lies within [A:B].

Note

Code for wxMaxima equation solve:

```
solve(  [a1=a*k+b, b1=b*k+b] ,[k,b] );
```

8.1.3.11.1 Implementation details First, find the linear coefficients `ck` and `cb` from the simple linear system.
Second, do the actual linear rescale of the input value.
Definition at line 5706 of file m_common.f90.

8.1.3.12 rescale_1()

```
elemental real(srp) function, private commondata::rescale_1 (
            real(srp), intent(in) value_in,
            real(srp), intent(in) A1,
            real(srp), intent(in) B1 )  [private]
```

Rescale a real variable with the range 0:1 to have the new range A1:B1.

Warning

The function does not check if `value_in` lies within [0:1].

Note

Code for wxMaxima equation solve:
`solve( [a1=0*k+b, b1=1*k+b] ,[k,b] );`

8.1.3.12.1 Implementation details First, find the linear coefficients `ck` and `cb` from the simple linear system. Second, do the actual linear rescale of the input value.
Definition at line 5734 of file m_common.f90.

8.1.3.13 within_r()

```
elemental real(srp) function, private commondata::within_r (
            real(srp), intent(in) value_in,
            real(srp), intent(in), optional vmin,
            real(srp), intent(in) vmax ) [private]
```

Force a value within the range set by the vmin and vmax dummy parameter values. If the value is within the range, it does not change, if it falls outside, the output force value is obtained as min(max(value, FORCE_MIN), FORCE_MAX)

Parameters

in	*value_in*	Input value for forcing transformation.
in	*vmin*	minimum value of the force-to range (lower limit), if not present, a lower limit of 0.0 is used.
in	*vmax*	maximum value of the force-to range (upper limit)

Returns

an input value forced to the range.

Note

Note that this is the **real** precision version of the generic `within` function.

Definition at line 5766 of file m_common.f90.

8.1.3.14 within_i()

```
elemental integer function, private commondata::within_i (
            integer, intent(in) value_in,
            integer, intent(in), optional vmin,
            integer, intent(in) vmax ) [private]
```

Force a value within the range set by the vmin and vmax dummy parameter values. If the value is within the range, it does not change, if it falls outside, the output force value is obtained as min(max(value, FORCE_MIN), FORCE_MAX)

Parameters

in	*value_in*	Input value for forcing transformation.
in	*vmin*	minimum value of the force-to range (lower limit), if not present, a lower limit of 0.0 is used.
in	*vmax*	maximum value of the force-to range (upper limit)

Returns

an input value forced to the range.

Note

Note that this is the **integer** version of the generic `within` function.

Definition at line 5798 of file m_common.f90.

8.1.3.15 is_within_r()

```
elemental logical function, private commondata::is_within_r (
            real(srp), intent(in) x,
            real(srp), intent(in) lower,
            real(srp), intent(in) upper )  [private]
```

Logical function to check if a value is within a specific range, **lower <= X <= upper**. The reverse (upper <= x <= lower) range limits can also be used; a corrective adjustment is automatically made.

Note

This is the real type version.

See commondata::is_within_operator_r() and commondata::is_within_operator_i() for a related user-defined operator `.within.`.

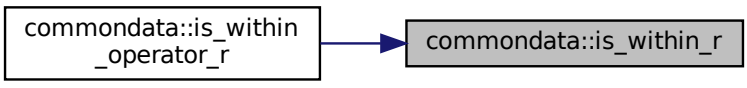
Parameters

in	*x*	the value to test
in	*lower*	the lower limit for the range tested.
in	*upper*	the upper limit of the range tested.

Returns

Returns TRUE if `x` lies within [lower,upper] and FALSE otherwise.

First, make sure the lower bound is actually smaller than the upper bound. If not, swapped bounds are used to set the range: [upper,lower].
Definition at line 5831 of file m_common.f90.
Here is the caller graph for this function:



8.1.3.16 is_within_i()

```
elemental logical function, private commondata::is_within_i (
            integer, intent(in) x,
            integer, intent(in) lower,
            integer, intent(in) upper )  [private]
```

Logical function to check if a value is within a specific range, **lower <= X <= upper**. The reverse (upper <= x <= lower) range limits can also be used; a corrective adjustment is automatically made.

Note

This is the integer type version.

See commondata::is_within_operator_r() and commondata::is_within_operator_i() for a related user-defined operator `.within.`.

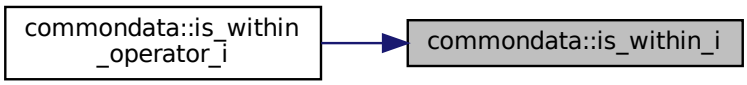
Parameters

in	*x*	the value to test
in	*lower*	the lower limit for the range tested.
in	*upper*	the upper limit of the range tested.

Returns

Returns TRUE if `x` lies within [lower,upper] and FALSE otherwise.

First, make sure the lower bound is actually smaller than the upper bound. If not, swapped bounds are used to set the range: [upper,lower].
Definition at line 5868 of file m_common.f90.
Here is the caller graph for this function:



8.1.3.17 is_within_operator_r()

```
pure logical function, private commondata::is_within_operator_r (
            real(srp), intent(in) x,
            real(srp), dimension(2), intent(in) limits )  [private]
```

A wrapper function for commondata::is_within() to build a user defined operator. Basically, it is the same as `is_within`, but the lower and upper limits are set as a two-element array. Usage of the operator:

```
if ( value .within. [lower, upper] ) then
```

Note

This is real type version of the procedure.

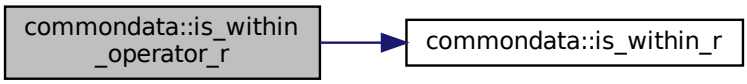
Parameters

in	*x*	x the value to test
in	*limits*	limits an array of two elements that sets the lower and upper limits for the range tested.

Returns

Returns TRUE if `x` lies within [lower,upper] and FALSE otherwise.

The implementation just calls the real type function commondata::is_within_r().
Definition at line 5902 of file m_common.f90.
Here is the call graph for this function:



8.1.3.18 is_within_operator_i()

```
pure logical function, private commondata::is_within_operator_i (
            integer, intent(in) x,
            integer, dimension(2), intent(in) limits )  [private]
```

A wrapper function for commondata::is_within() to build a user defined operator. Basically, it is the same as `is_within`, but the lower and upper limits are set as a two-element array. Usage of the operator:

```
if ( value .within. [lower, upper] ) then
```

Note

This is an integer type version of the procedure.

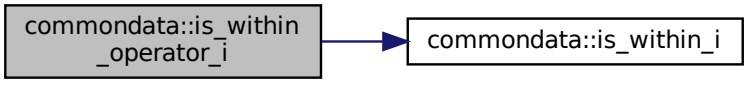
Parameters

in	*x*	x the value to test
in	*limits*	limits an array of two elements that sets the lower and upper limits for the range tested.

Returns

Returns TRUE if `x` lies within [lower,upper] and FALSE otherwise.

The implementation just calls the integer type function commondata::is_within_i().
Definition at line 5927 of file m_common.f90.
Here is the call graph for this function:



8.1.3.19 average_r()

```
pure real(srp) function, private commondata::average_r (
            real(srp), dimension(:), intent(in) array_in,
            real(srp), intent(in), optional missing_code,
            logical, intent(in), optional undef_ret_null )  [private]
```

Calculate an average value of a real array, excluding MISSING values.

Parameters

vector_in	The input data vector
missing_code	Optional parameter setting the missing data code, to be excluded from the calculation of the mean.
undef_ret_null	Optional parameter, if TRUE, the function returns zero rather than undefined if the sample size is zero.

Returns

The mean value of the vector.

Note

This is a real array version.

8.1.3.19.1 Implementation details Check if missing data code is provided from dummy input. If not, use global parameter.
Fist, count how many valid values are there in the array.
If there are no valid values in the array, mean is undefined.
still return zero if undef_ret_null is TRUE.
Definition at line 5952 of file m_common.f90.

8.1.3.20 average_i()

```
pure real(srp) function, private commondata::average_i (
            integer, dimension(:), intent(in) array_in,
            integer, intent(in), optional missing_code,
            logical, intent(in), optional undef_ret_null )  [private]
```

Calculate an average value of an integer array, excluding MISSING values.

Returns

The mean value of the vector

Parameters

vector_in	The input data vector
missing_code	Optional parameter setting the missing data code, to be excluded from the calculation of the mean.
undef_ret_null	Optional parameter, if TRUE, the function returns zero rather than undefined if the sample size is zero.

Note

This is an integer array version.

8.1.3.20.1 Implementation details Check if missing data code is provided from dummy input. If not, use global parameter.

Fist, count how many valid values are there in the array.
If there are no valid values in the array, mean is undefined.
still return zero if undef_ret_null is TRUE.
Definition at line 6022 of file m_common.f90.

8.1.3.21 std_dev()

```
real(srp) function commondata::std_dev (
             real(srp), dimension(:), intent(in) array_in,
             real(srp), intent(in), optional missing_code,
             logical, intent(in), optional undef_ret_null )
```

Calculate standard deviation using trivial formula:

$$\sigma = \sqrt{\frac{\sum(x - \overline{x})^2}{N - 1}}.$$

Note

This is a real array version.

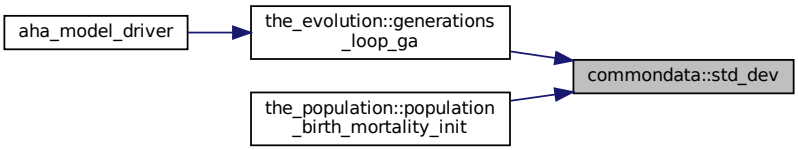
Parameters

in	*array_in*	vector_in The input data vector
in	*missing_code*	missing_code Optional parameter setting the missing data code, to be excluded from the calculation of the mean.
in	*undef_ret_null*	undef_ret_null Optional parameter, if TRUE, the function returns zero rather than undefined if the sample size is zero.

Returns

The standard deviation of the data vector.

Check if missing data code is provided from dummy input. If not, use global parameter.
If there are no valid values in the array, std. dev. is undefined.
still return zero if undef_ret_null is TRUE.
Definition at line 6088 of file m_common.f90.
Here is the caller graph for this function:



8.1.3.22 stack2arrays_r()

```
pure real(srp) function, dimension(:), allocatable, private commondata::stack2arrays_r (
             real(srp), dimension(:), intent(in) a,
             real(srp), dimension(:), intent(in) b )  [private]
```

Concatenate two arrays a and b. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.

Note

This is the **real** type version.

Parameters

in	*a*	param[in] a first array
in	*b*	param[in] b second array

Returns

return an array [a, b]

Definition at line 6156 of file m_common.f90.

8.1.3.23 stack2arrays_i()

```
pure integer function, dimension(:), allocatable, private commondata::stack2arrays_i (
            integer, dimension(:), intent(in) a,
            integer, dimension(:), intent(in) b )  [private]
```

Concatenate two arrays a and b. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.

Note

This is the **real** type version.

Parameters

in	*a*	param[in] a first array
in	*b*	param[in] b second array

Returns

return an array [a, b]

Definition at line 6175 of file m_common.f90.

8.1.3.24 is_near_zero_srp()

```
elemental logical function, private commondata::is_near_zero_srp (
            real(srp), intent(in) test_number,
            real(srp), intent(in), optional epsilon )  [private]
```

Checks if a real number is near 0.0. Thus function can be used for comparing two real values like the below.

```
if ( is_near_zero(a) ) then ...
```

Note

Note that commondata::float_equal() can be used for approximate real comparisons of two real type values.

Modified from `Near0_dp()` function from Clerman & Spector 2012, p. 250-251.

This is the **standard** precision function (commondata::srp).

Parameters

in	*test_number*	test_number the number to check for being near-zero.
in	*epsilon*	epsilon optional (very small) tolerance value.

Returns

TRUE if the `test_number` is near-zero.

Definition at line 6202 of file m_common.f90.

8.1.3.25 is_near_zero_hrp()

```
elemental logical function, private commondata::is_near_zero_hrp (
             real(hrp), intent(in) test_number,
             real(hrp), intent(in), optional epsilon )  [private]
```

Checks if a real number is near 0.0. Thus function can be used for comparing two real values like the below.

```
if ( is_near_zero(a) ) then ...
```

Note

Note that commondata::float_equal() can be used for approximate real comparisons of two real type values.

Modified from `Near0_dp()` function from Clerman & Spector 2012, p. 250-251.

This is the **high** precision function (commondata::hrp).

Parameters

in	*test_number*	test_number the number to check for being near-zero.
in	*epsilon*	epsilon optional (very small) tolerance value.

Returns

TRUE if the `test_number` is near-zero.

Definition at line 6232 of file m_common.f90.

8.1.3.26 float_equal_srp()

```
elemental logical function, private commondata::float_equal_srp (
             real(srp), intent(in) value1,
             real(srp), intent(in) value2,
             real(srp), intent(in), optional epsilon )  [private]
```

Check if two real values are nearly equal using the commondata::is_near_zero(). Thus function can be used for comparing two real values like the below. The exact comparison (incorrect due to possible rounding):

```
if ( a == b ) ...
```

should be substituted by such comparison:

```
if ( float_equal(a, b, 0.00001) ) ...
```

There is also a user defined operator `.feq.` for approximate float point equality. It differs from this function in that the later allows to set an arbitrary epsilon tolerance value whereas the operator does not (it uses the default epsilon based on the intrinsic `tiny()` function.

Note

This is the **standard** precision function (commondata::srp).

Parameters

in	*value1*	value1 The first value for comparison.
in	*value2*	value2 The second value for comparison.
in	*epsilon*	epsilon optional (very small) tolerance value.

Returns

TRUE if the values `value1` and `value2` are nearly equal.

Definition at line 6268 of file m_common.f90.

8.1.3.27 float_equal_hrp()

```
elemental logical function, private commondata::float_equal_hrp (
             real(hrp), intent(in) value1,
             real(hrp), intent(in) value2,
             real(hrp), intent(in), optional epsilon )  [private]
```

Check if two real values are nearly equal using the commondata::is_near_zero(). Thus function can be used for comparing two real values like the below. The exact comparison (incorrect due to possible rounding):

```
if ( a == b ) ...
```

should be substituted by such comparison:

```
if ( float_equal(a, b, 0.00001) ) ...
```

There is also a user defined operator `.feq.` for approximate float point equality. It differs from this function in that the later allows to set an arbitrary epsilon tolerance value whereas the operator does not (it uses the default epsilon based on the intrinsic `tiny()` function.

Note

This is the **high** precision function (commondata::hrp).

Parameters

in	*value1*	value1 The first value for comparison.
in	*value2*	value2 The second value for comparison.
in	*epsilon*	epsilon optional (very small) tolerance value.

Returns

TRUE if the values `value1` and `value2` are nearly equal.

Definition at line 6306 of file m_common.f90.

8.1.3.28 float_equal_srp_operator()

```
elemental logical function, private commondata::float_equal_srp_operator (
             real(srp), intent(in) value1,
             real(srp), intent(in) value2 )  [private]
```

This is a wrapper for the commondata::float_equal_srp() for building the user defined operator `.feq.` with default tolerance (`epsilon` parameter). The exact real comparison (incorrect due to possible rounding):

```
if ( a == b ) ...
```

should be substituted by such comparison:

```
if ( a .feq. b) ...
```

Warning

This function is not intended for direct use.

Parameters

in	*value1*	value1 The first value for comparison.
in	*value2*	value2 The second value for comparison.

Returns

TRUE if the values `value1` and `value2` are nearly equal.

Definition at line 6340 of file m_common.f90.

8.1.3.29 float_equal_hrp_operator()

```
elemental logical function, private commondata::float_equal_hrp_operator (
            real(hrp), intent(in) value1,
            real(hrp), intent(in) value2 )  [private]
```

This is a wrapper for the commondata::float_equal_hrp() for building the user defined operator `.feq.` with default tolerance (`epsilon` parameter). The exact real comparison (incorrect due to possible rounding):

```
if ( a == b ) ...
```

should be substituted by such comparison:

```
if ( a .feq. b) ...
```

Warning

This function is not intended for direct use.

Parameters

in	*value1*	value1 The first value for comparison.
in	*value2*	value2 The second value for comparison.

Returns

TRUE if the values `value1` and `value2` are nearly equal.

Definition at line 6365 of file m_common.f90.

8.1.3.30 float_approx_srp_operator()

```
elemental logical function, private commondata::float_approx_srp_operator (
            real(srp), intent(in) value1,
            real(srp), intent(in) value2 )  [private]
```

This is a wrapper for the commondata::float_equal_srp() for building the user defined operator `.approx.` with **very high** tolerance (`epsilon` parameter). The exact real comparison (incorrect due to possible rounding):

```
if ( a == b ) ...
```

should be substituted by such comparison:

```
if ( a .approx. b) ...
```

Warning

This function is not intended for direct use.

Parameters

in	*value1*	value1 The first value for comparison.
in	*value2*	value2 The second value for comparison.

Returns

TRUE if the values `value1` and `value2` are approximately equal.

Definition at line 6390 of file m_common.f90.

8.1.3.31 float_approx_hrp_operator()

```
elemental logical function, private commondata::float_approx_hrp_operator (
            real(hrp), intent(in) value1,
            real(hrp), intent(in) value2 )  [private]
```

This is a wrapper for the commondata::float_equal_hrp() for building the user defined operator `.approx.` with **very high** tolerance (`epsilon` parameter). The exact real comparison (incorrect due to possible rounding):

```
if ( a == b ) ...
```

should be substituted by such comparison:

```
if ( a .approx. b) ...
```

Warning

This function is not intended for direct use.

Parameters

in	*value1*	value1 The first value for comparison.
in	*value2*	value2 The second value for comparison.

Returns

TRUE if the values `value1` and `value2` are approximately equal.

Definition at line 6416 of file m_common.f90.

8.1.3.32 do_sanitise()

```
subroutine commondata::do_sanitise (
            real(srp), dimension(:), intent(inout) array,
            real(srp), intent(in), optional lvalid,
            real(srp), intent(in), optional hvalid,
            real(srp), intent(in), optional substval,
            logical, intent(in), optional only_wrong,
            character(len=*), intent(in), optional tnote )
```

Sanitize a real commondata::srp array, so that any value that is smaller than the minimum sensible value `lvalid` or greater then the maximum sensible value `hvalid` is substituted with `substval`. The procedure also checks the input value for IEEE validity: overflow, underflow, invalid and inexact.

Note

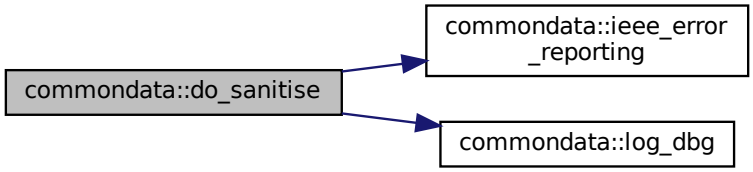
Note that commondata::do_sanitise() makes use the IEEE math procedures to handle halting on invalid values.

Parameters

in,out	*array*	[inout] array the array to be sanitised
in	*lvalid*	Optional lower valid boundary, default is set to commondata::lo_valid_sanitised
in	*hvalid*	Optional higher valid boundary, default is set to commondata::hi_valid_sanitised
in	*substval*	Optional substitution value, default is commondata::missing
in	*only_wrong*	Optional flag to correct only invalid values (`Nan`, `Inf`) without range correction (`lvalid`, `hvalid`), default value is FALSE, i.e. to correct for range (`lvalid` to `hvalid`)
in	*tnote*	Optional text note that is printed in case of IEEE error signalling to assist in the identification of the specific place in the code.

The input array is checked for wrong values: `NaN`, `Inf+` and `Inf-`. Any invalid values are substituted by the `substval` or missing if that is not defined.

Definition at line 6437 of file m_common.f90.
Here is the call graph for this function:



8.1.3.33 ieee_error_reporting()

```
integer function commondata::ieee_error_reporting (
            logical, intent(in), optional reset,
            character(len=*), intent(in), optional tnote )
```

Check if an IEEE error condition has occurred.

Parameters

in	*reset*	Logical flag requesting to reset any exceptions that are found (if set to TRUE), default value is TRUE. IEEE exception condition will not be reset if `reset` is `FALSE`.

Returns

integer IEEE Error code or null if there is no error condition.

```
IS_OVERFLOW=101,
IS_UNDERFLOW=102,
IS_INVALID=103,
IS_INEXACT=104,
IS_DIVBYZERO=105,
NO_SUPPORT_OVERFLOW=901,
NO_SUPPORT_UNDERFLOW=902,
NO_SUPPORT_INVALID=903,
NO_SUPPORT_INEXACT=904,
NO_SUPPORT_DIVBYZERO=905,
NO_ERROR_FLAG=0
```

Implementation notes ### The automatically generated IEEE wrapper `include "IEEE_wrap.inc` is used to invoke intrinsic or non-intrinsic IEEE math modules. If the compiler support intrinsic modules, this can be changed to

```
use, intrinsic :: ieee_arithmetic
use, intrinsic :: ieee_exceptions
```

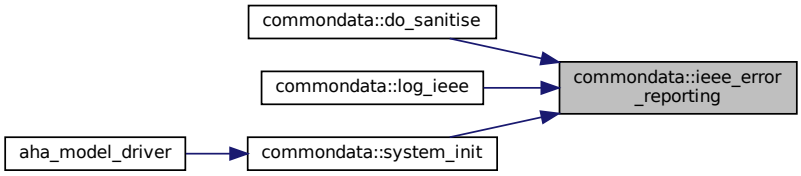
Warning

Note that IEEE routines cannot be used in pure procedures.

Parameters

in	*tnote*	Optional text note that is printed in case of IEEE error signalling to assist in the identification of the specific place in the code.

Definition at line 6583 of file m_common.f90.
Here is the caller graph for this function:



8.1.3.34 zeroin()

```
real(srp) function commondata::zeroin (
            real(srp), intent(in) ax,
            real(srp), intent(in) bx,
            real(srp) f,
            real(srp), intent(in) tol )
```

This function calculates a zero of a function f(x) in the interval (ax,bx).

- Brent, R.P., (1973). Algorithms for Minimization Without Derivatives, Prentice-Hall, Inc.
- Brent, R.P. (1971). An algorithm with guaranteed convergence for finding a zero of a function, Computer J. 14, 422–425.

Author: Richard Brent, https://maths-people.anu.edu.au/~brent/ Source: http://www.netlib.org/go/ With some minor changes by Sergey Budaev.

Note

This function is used in the_environment::minimum_depth_visibility().

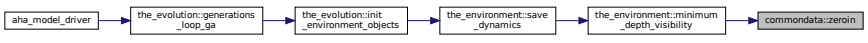
Parameters

in	*ax*	left endpoint of initial interval
in	*bx*	right endpoint of initial interval
in	*f*	function subprogram which evaluates f(x) for any x in the interval (ax,bx).
in	*tol*	desired length of the interval of uncertainty of the final result (.ge.0.)

Returns

Abscissa approximating a zero of f(x) in the interval (ax,bx). Note that this function returns commondata::missing if f(ax) and f(bx) do not have different signs (so there is no function zero within the range).

Definition at line 6744 of file m_common.f90.
Here is the caller graph for this function:



8.1.3.35 allelescale()

```
elemental real(srp) function commondata::allelescale (
            integer, intent(in) raw_value )
```

Converts and rescales integer allele value to real value for neural response function.

Parameters

in	*raw_value*	raw input value, integer within commondata::allelerange_min and commondata::allelerange_max

Return values

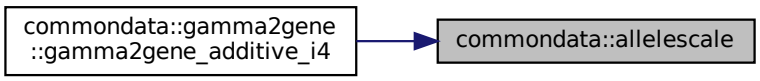
Returns	the value of conversion function: integer alleles to real internal value zero to allelescale_max

Allele conversion function for the relationship between the genome integer allele value and its converted real value in the neuronal response function. The function rescales integer allele value I_i within the range $[I_{min}, I_{max}]$ to real values r within the range $[0, M]$, where M is defined by the allelescale_max parameter. Conversion is performed by the ::rescale() backend function. See implementation notes on `allele_value` component of the `GENE` derived type.

Note

Note that it is an *elemental function* that can accept both scalar and vector arguments.

Definition at line 6882 of file m_common.f90.
Here is the caller graph for this function:



8.1.3.36 alleleconv()

```
elemental real(srp) function commondata::alleleconv (
            real(srp), intent(in) raw_value )
```

Rescales the relationship between the numerical value of an allele in the genome and the numerical value to be used in the neuronal response function.

Parameters

raw_value	Raw input value from the genome.

Return values

Returns	the rescaled value.

Note

This is the same component as in the original model ALLELECONV parameter. Thus, if the allele value is integer in `gamma2gene`, then `commondata::allelescale()` is called first (convert integer to real 0..1), then follows `alleleconv()` (rescale 0..1 to rescaled value).

Note that it is an *elemental function* that can accept both scalar and vector arguments.

Scale factor for the simple linear conversion (Type 3).

8.1.3.36.1 Implementation details

8.1.3.36.2 Type 1 *Type 1:* no conversion from 0:1 to output allele value

Note

identical to old alleleconv 1

```
converted = raw_value
```

8.1.3.36.3 Type 2 *Type 2:* exponential conversion from 0:1 to output allele value, to allow finer resolution at weak perception strengths.

Warning

This version can produce NaNs.

Note

identical to identical to old alleleconv 2

```
converted = (10 * raw_value) ** 2
```

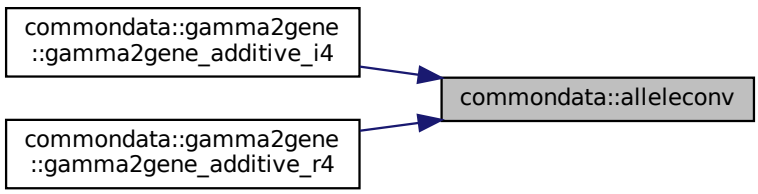
8.1.3.36.4 Type 3 *Type 3:* linear conversion from 0:1 to output allele value, has been the most used value, although not perfect.

Note

identical to old alleleconv 3

```
converted = raw_value * SCALE_FACTOR
```

Definition at line 6906 of file m_common.f90.
Here is the caller graph for this function:



8.1.3.37 cv2variance()

```
elemental real(srp) function commondata::cv2variance (
            real(srp), intent(in) cv,
            real(srp), intent(in) mean )
```

Calculate the variance from the coefficient of variation.

The coefficient of variation `cv` $cv = \frac{\sigma}{\bar{P}}$. Therefore, the raw variance σ^2 in the `RNORM` function is equal to

$$\sigma^2 = (cv \cdot \bar{P})^2.$$

Returns

Variance σ^2

Parameters

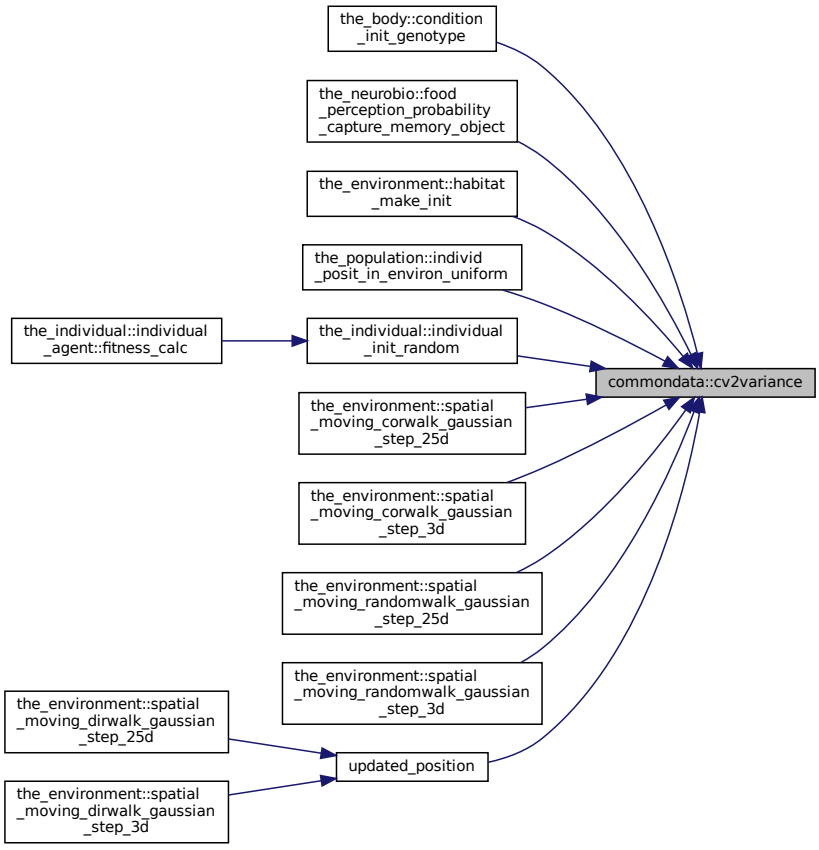
cv	Coefficient of variation.
mean	Average.

Note

We need this function because Gaussian parameters in `commondata` are set using the *average* and *CV* to be more intuitive, scale independent. But the function `RNORM` in `HEDTOOLS` gets variance as a parameter.

Note that this is an elemental function that gets both array and scalar parameters.

Definition at line 6955 of file m_common.f90.
Here is the caller graph for this function:



8.1.3.38 gamma2gene_additive_i4()

```
real(srp) function, private commondata::gamma2gene_additive_i4 (
            integer, dimension(:), intent(in) gs,
            integer, dimension(:), intent(in) gh,
            real(srp), intent(in) signal,
            real(srp), intent(in), optional erpcv )  [private]
```

The function gamma2gene finds the sigmoid relationship for a complex multicomponent 2-allele impact on the neuronal response.
The real function gamma2gene finds the sigmoid relationship for a multicomponent allele impact on the neuronal response:

$$R = \frac{(P/y_1)^{x_1}}{1+(P/y_1)^{x_1}} + \frac{(P/y_2)^{x_2}}{1+(P/y_2)^{x_2}} + \frac{(P/y_3)^{x_3}}{1+(P/y_3)^{x_3}} ...$$

Here, R is the neuronal response, P the strength of the sensory input (scaled 0-1), and x and y are two genes. The indices refer to the additive components of the alleles. Note that their number is set by the parameter ADDITIVE↩ _COMPS. Further, erpcv defines the coefficient of variation for the perception error (with respect to its true value).

Returns

returns the neuronal response.

Parameters

in	*gs*	shape: Gene/constant determining the shape of the gamma function. Note that the raw integer gene values are accepted by this function as commondata::allelescale() is called automatically inside.
in	*gh*	half-max effect: Gene/constant for the signal strength giving half max effect. Note that the raw integer gene values are accepted by this function as commondata::allelescale() is called automatically inside.
in	*signal*	perception: Input value of (external or internal) stimulus perception.
in	*erpcv*	error: Additive error of stimulus perception, Gaussian variance added to the true environmental variable. If this parameter is absent, no perception error is introduced. Maxima function for quick calc: `g2gene(p,x,y,n) := n * ( (p/y)^x / (1+(p/y)^x) );`

Warning

This version of gamma2gene accepts *integer* arrays. It *does* invoke commondata::allelescale() automatically inside.

8.1.3.38.1 Implementation details Perception is calculated as

$$P = \bar{P} + \varepsilon,$$

where $\bar{P}$ is the true environmental variable and ε is a Gaussian error. The perception value with error is implemented as a normal Gaussian variate with the mean equal to the true signal value $\bar{P}$ and the coefficient of variation equal to the erpcv input parameter: $erpcv = \frac{\sigma}{\bar{P}}$. Therefore, the raw error variance in the RNORM function is equal to the square of erpcv*signal. We also impose strict limit on perception $P > 0$.
Definition at line 7003 of file m_common.f90.

8.1.3.39 gamma2gene_additive_r4()

```
real(srp) function, private commondata::gamma2gene_additive_r4 (
            real(srp), dimension(:), intent(in) gs,
            real(srp), dimension(:), intent(in) gh,
            real(srp), intent(in) signal,
            real(srp), intent(in), optional erpcv )  [private]
```

The function gamma2gene finds the sigmoid relationship for a complex multicomponent 2-allele impact on the neuronal response.

Returns

returns the neuronal response.

Parameters

in	*shape*	Gene/constant determining the shape of the gamma function. Note that the raw integer gene values are accepted by this function as commondata::allelescale() is called automatically inside.
in	*half-max*	effect: Gene/constant for the signal strength giving half max effect. Note that the raw integer gene values are accepted by this function as commondata::allelescale() is called automatically inside.
in	*perception*	Input value of (external or internal) stimulus perception.
in	*error*	Additive error of stimulus perception, Gaussian variance added to the true environmental variable. If this parameter is absent, no perception error is introduced.

The real function gamma2gene finds the sigmoid relationship for a multicomponent allele impact on the neuronal response:

$$R = \frac{(P/y_1)^{x_1}}{1+(P/y_1)^{x_1}} + \frac{(P/y_2)^{x_2}}{1+(P/y_2)^{x_2}} + \frac{(P/y_3)^{x_3}}{1+(P/y_3)^{x_3}} ...$$

Here, R is the neuronal response, P the strength of the sensory input (scaled 0-1), and x and y are two genes. The indices refer to the additive components of the alleles. Note that their number is set by the parameter ADDITIVE_COMPS. Further, erpcv defines the coefficient of variation for the perception error (with respect to its true value).
Maxima function for quick calc:

```
g2gene(p,x,y,n) := n * ( (p/y)^x / (1+(p/y)^x) );
```

Warning

This version of gamma2gene accepts *real* arrays. It does *not* invoke commondata::allelescale() automatically inside.

Perception is calculated as

$$P = \bar{P} + \varepsilon,$$

where $\bar{P}$ is the true environmental variable and ε is a Gaussian error. The perception value with error is implemented as a normal Gaussian variate with the mean equal to the true signal value $\bar{P}$ and the coefficient of variation equal to the erpcv input parameter: $erpcv = \frac{\sigma}{\bar{P}}$. Therefore, the raw error variance in the RNORM function is equal to the square of erpcv*signal. We also impose strict limit on perception min=0.
Definition at line 7115 of file m_common.f90.

8.1.3.40 gamma2gene_fake_vals()

```
elemental real(srp) function, private commondata::gamma2gene_fake_vals (
            real(srp), intent(in) signal,
            real(srp), intent(in), optional gs,
            real(srp), intent(in), optional gh,
            integer, intent(in), optional n_acomps )  [private]
```

This "fake" version of the gamma2gene is used to guess the response values in calculations.

Return values

predicted_val	a predicted value (scalar or array) of the sigmoidal neuronal response function. See gamma2gene for details.

Definition at line 7193 of file m_common.f90.

8.1.3.41 gamma2gene_reverse()

```
elemental real(srp) function, private commondata::gamma2gene_reverse (
            real(srp), intent(in) neuronal_response,
            real(srp), intent(in) gs,
            real(srp), intent(in) gh,
            integer, intent(in), optional nc )  [private]
```

Reverse-calculate perception value from the given neural response value.
Calculates the value of the raw perception from the neural response function. This is the reverse of the gamma2gene with many components. It is assumed that all x and y values are the same, so the equation solved for the most trivial case. Calculated according to the formula:

$$P = y \left(\frac{R}{n - R} \right)^{1/x},$$

where P is the perception value, R is the neural response, x and y are two genes.

Returns

Signal level for specific neural response, back calculated.

Parameters

in	*neuronal_response*	neuronal response.
in	*gs*	shape parameter of the sigmoid function.
in	*gh*	half-max parameter of the sigmoid function.
in	*nc*	Number of additive components. Optional, if absent assumed 1 (single component).

Note

This function is useful for guessing the average start values of genetically determined traits with Gaussian distribution.

Note that it is quite difficult to get really small `gamma2gene` values as the signal value should be really small: e.g. to get neural response 1.5E-5 (Fulton condition), we need signal = 2E-12. So, the function very quickly loses precision as we approach really low values. Need kind 8 or 16 precision?

Warning

This is quite a crude guess at low values. At lower values *underestimates* R, real value is higher. This is due to the limitation that R should never be below zero, causing above-zero truncation.

Returns

Signal level for specific neural response, back calculated.

Maxima function for quick calc:

```
reverse_gamma(r,x,y,n) := y * (r/(n-r))^(1/x);
```

Definition at line 7256 of file m_common.f90.

8.1.3.42 add_to_history_i4()

```
pure subroutine, private commondata::add_to_history_i4 (
            integer, dimension(:), intent(inout) history_array,
            integer, intent(in) add_this )  [private]
```

Simple history stack function, add to the end of the stack. We need only to add components on top of the stack and retain commondata::history_size_spatial elements of the prior history (for a spatial moving object). The stack works as follows, assuming 100 and 200 are added:
[1 2 3 4 5 6 7 8 9 10];
[2 3 4 5 6 7 8 9 10 100];
[3 4 5 6 7 8 9 10 100 200].

Parameters

history_array	Integer array that keeps the history.
add_this	we add this element to the end of the history array.

Note

This is the integer type version.

Definition at line 7299 of file m_common.f90.

8.1.3.43 add_to_history_r()

```
pure subroutine, private commondata::add_to_history_r (
            real(srp), dimension(:), intent(inout) history_array,
            real(srp), intent(in) add_this )  [private]
```

Simple history stack function, add to the end of the stack. We need only to add components on top of the stack and retain commondata::history_size_spatial elements of the prior history (for a spatial moving object).

Parameters

history_array	Integer array that keeps the history.
add_this	we add this element to the end of the history array.

Note

This is the real type version.

Definition at line 7324 of file m_common.f90.

8.1.3.44 add_to_history_char()

```
pure subroutine, private commondata::add_to_history_char (
            character(*), dimension(:), intent(inout) history_array,
            character(*), intent(in) add_this )  [private]
```

Simple history stack function, add to the end of the stack. We need only to add components on top of the stack and retain commondata::history_size_spatial elements of the prior history.

Parameters

history_array	Integer array that keeps the history.
add_this	we add this element to the end of the history array.

Note

This is the character string type version

Definition at line 7348 of file m_common.f90.

8.1.3.45 conv_l2i()

```
elemental integer function commondata::conv_l2i (
            logical, intent(in) flag,
            integer, intent(in), optional code_false,
            integer, intent(in), optional code_true )
```

Converts logical to integer following a rule, default FALSE = 0, TRUE = 1.

Note

Note that this function is required to place logical data like the survival status (alive) into the reshape array that is saved as the CSV data file.

Definition at line 7370 of file m_common.f90.

8.1.3.46 conv_l2r()

```
elemental real(srp) function commondata::conv_l2r (
            logical, intent(in) flag,
            real(srp), intent(in), optional code_false,
            real(srp), intent(in), optional code_true )
```

Converts logical to standard (kind SRP) real, .FALSE. => 0, .TRUE. => 1.

Note

Note that this function is required to place logical data like the survival status (alive) into the reshape array that is saved as the CSV data file. **Example: **

```
call CSV_MATRIX_WRITE ( reshape(                              &
           [ habitat_safe%food%food%x,                        &
             habitat_safe%food%food%y,                        &
             habitat_safe%food%food%depth,                    &
             conv_l2r(habitat_safe%food%food%eaten),          &
             habitat_safe%food%food%size],                    &
           [habitat_safe%food%number_food_items, 5]),         &
           "zzz_food_s" // model_name // "_" // mmdd //       &
             "_gen_" // tostr(generat, generations) // csv,   &
           ["X   ","Y   ", "D   ", "EATN", "SIZE"]           &
           )
```

First, check optional parameters.
Definition at line 7420 of file m_common.f90.

8.1.3.47 is_maxval_r()

```
pure logical function, private commondata::is_maxval_r (
            real(srp), intent(in) value,
            real(srp), dimension(:), intent(in) array,
            real(srp), intent(in), optional tolerance )  [private]
```

Function to check if the value is the maximum value of an array (returns TRUE), or not (return FALSE).

Returns

TRUE if `value` is indeed the maximum value of the `array` and FALSE otherwise.

Parameters

in	*value*	The value to check
in	*array*	The array to check within.
in	*tolerance*	Optional tolerance threshold.

Note

Check if we are provided with the tolerance threshold, if not, use the default parameter commondata::tolerance_low_def_srp value.

Definition at line 7459 of file m_common.f90.

8.1.3.48 is_maxval_i()

```
pure logical function, private commondata::is_maxval_i (
            integer, intent(in) value,
            integer, dimension(:), intent(in) array )  [private]
```

Function to check if the value is the maximum value of an array (returns TRUE), or not (return FALSE). Integer version.

Returns

TRUE if `value` is indeed the maximum value of the `array` and FALSE otherwise.

Parameters

in	*value*	The value to check
in	*array*	The array to check within.

Definition at line 7497 of file m_common.f90.

8.1.3.49 is_minval_r()

```
pure logical function, private commondata::is_minval_r (
            real(srp), intent(in) value,
            real(srp), dimension(:), intent(in) array,
            real(srp), intent(in), optional tolerance )  [private]
```

Function to check if the value is the minimum value of an array (returns TRUE), or not (return FALSE).

Returns

TRUE if `value` is indeed the minimum value of the `array` and FALSE otherwise.

Parameters

in	*value*	The value to check
in	*array*	The array to check within.
in	*tolerance*	Optional tolerance threshold.

Note

Check if we are provided with the tolerance threshold, if not, use the default parameter commondata::tolerance_low_def_srp value.

Definition at line 7524 of file m_common.f90.

8.1.3.50 is_minval_i()

```
pure logical function, private commondata::is_minval_i (
            integer, intent(in) value,
            integer, dimension(:), intent(in) array )  [private]
```

Function to check if the value is the minimum value of an array (returns TRUE), or not (return FALSE). Integer version.

Returns

TRUE if `value` is indeed the minimum value of the `array` and FALSE otherwise.

Parameters

in	*value*	The value to check
in	*array*	The array to check within.

Definition at line 7562 of file m_common.f90.

8.1.3.51 timer_cpu_start()

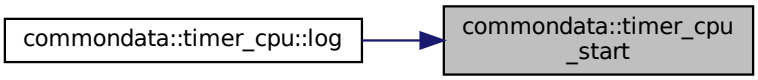
```
subroutine, private commondata::timer_cpu_start (
             class(timer_cpu), intent(inout) this,
             character(len=*), intent(in), optional timer_title )  [private]
```

Start the timer object, stopwatch is now ON.

Parameters

title,an	optional title for the stopwatch object

Definition at line 7589 of file m_common.f90.
Here is the caller graph for this function:



8.1.3.52 timer_cpu_elapsed()

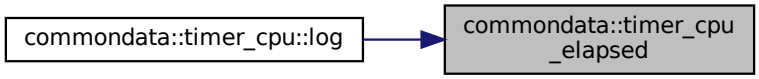
```
real(srp) function, private commondata::timer_cpu_elapsed (
             class(timer_cpu), intent(in) this )  [private]
```

Calculate the time elapsed since the stopwatch subroutine was called for this instance of the timer container object. Can be called several times showing elapsed time since the grand start.

Returns

the time elapsed since `timer_cpu_start` call (object-bound).

Definition at line 7611 of file m_common.f90.
Here is the caller graph for this function:



8.1.3.53 timer_cpu_title()

```
character(len=:)  function, allocatable, private commondata::timer_cpu_title (
            class(timer_cpu), intent(in) this )  [private]
```

Return the title of the current timer object.

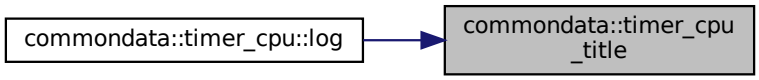
Returns

title of the stopwatch timer, allocatable with automatic trimming.

Note

Useful for outputs.

Definition at line 7632 of file m_common.f90.
Here is the caller graph for this function:



8.1.3.54 timer_cpu_show()

```
character(len=:)  function, allocatable, private commondata::timer_cpu_show (
            class(timer_cpu), intent(in) this )  [private]
```

A ready to use in output function that returns a formatted string for a timer combining its title and the elapsed time.
For example: `Calculating decomposition took 20s`.

Note

If the value in *seconds* is too big, hours passed are appended in parentheses.

Definition at line 7647 of file m_common.f90.
Here is the caller graph for this function:

8.1.3.55 timer_cpu_log()

```
character(len=:)  function, allocatable, private commondata::timer_cpu_log (
            class(timer_cpu) this )  [private]
```

A ready to use shortcut function to be used in logger, just adds the TIMER: tag in front of the normal `show`output.
Example use:

```
call stopwatch_1%start("Calculate matrix decomposition")
.....
call LOG_DBG( stopwatch_1%log() )
```

Definition at line 7679 of file m_common.f90.
Here is the caller graph for this function:

8.1.3.56 call_external()

```
subroutine commondata::call_external (
            character(len=*), intent(in) command,
            logical, intent(in), optional suppress_output,
            logical, intent(in), optional suppress_error,
            logical, intent(in), optional is_background_task,
            logical, intent(out), optional cmd_is_success,
            integer, intent(out), optional exit_code )
```

Call an external program using a command line. Wrapper to two alternative system shell calling intrinsic procedures.

Parameters

in	*command*	is the command line that should be run by the system.
in	*suppress_output*	is an optional logical flag indicating whether to suppress any STDOUT output (silent mode). Default is FALSE, i.e. do show command output (verbose).
in	*suppress_error*	is a similar optional flag to suppress STDERR reporting (silent mode). Default is FALSE.

Parameters

in	*is_background_task*	is a logical flag to set execution on the background, based on wait=.false. parameter of EXECUTE_COMMAND_LINE. The **default** value is TRUE, i.e. **do** process the command at the background. **Warning:** Setting command execution as a background task will make cmd_is_success unusable because the exit code of the child process is is deferred.
out	*cmd_is_success*	is a logical flag indicating the command result was *success* (zero exit code).
out	*exit_code*	exit code of the child process.

Warning

All external calls should normally be performed using this wrapper.

8.1.3.56.1 Implementation details Output is suppressed by redirection to the null device, which is platform-specific.
Output of the STDERR is also redirected to the platform-specific null device. For the Windows platform STDERR redirection see http://support.microsoft.com/en-us/kb/110930.
Check background task optional option is_background_task.

Note

If the older non-standard system (disabled) command is used for executing background task on Unix systems may add "&" at the end of the command, but cannot be easily implemented on Windows. This functionality is currently not implemented as system is disabled.

Background parallel execution is enabled by default.
The intrinsic procedure system is a GNU extension and may not be available on all platforms and compilers. It is currently disabled.

Note

If the older non-standard system command is used for executing background task on Unix systems may add "&" at the end of the command, but cannot be easily implemented on Windows. This functionality is currently not implemented as system is normally disabled. A further caveat with system is that the returned integer exit status parameter works only using the IFPORT portability library on Intel Fortran.

Warning

The system is normally **disabled**. Should be enabled if the standard-compliant execute_command_↩ line is not supported by the local compiler or results in errors.

On some systems, notably 'ahaworkshop', execute_command_line issues runtime error: Could not execute command line when calling the debug plotting utilities

- commondata::debug_histogram_save()
- commondata::debug_scatterplot_save()
- commondata::debug_interpolate_plot_save()

The reason of such crashes is unknown. Probably this caused by system specific limitations on child processes when numerous child processes of plotting are generated too quickly. A workaround is to disable debug plots by setting the parameter commondata::is_plotting FALSE using the environment variable AHA_DEBUG_PLOTS=NO. If plotting is absolutely essential, execute_command_line should be disabled and the inferior system call used. The latter does not seem to result in crashes, although this has not been well tested.

```
call system( cmd_execute, status_int )  ! GNU Fortran
call system( cmd_execute )              ! Intel Fortran requires IFPORT
```

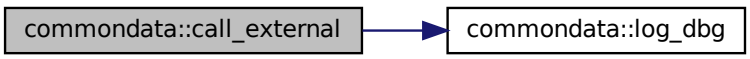
The F2008 execute_command_line should be better used here because it allows to control asynchronous/synchronous command execution and returns the exit status.

```
call execute_command_line( command  = cmd_execute,                  &
                           exitstat = status_int, wait = wait_exec,  &
                           cmdmsg = cmd_error_msg )
```

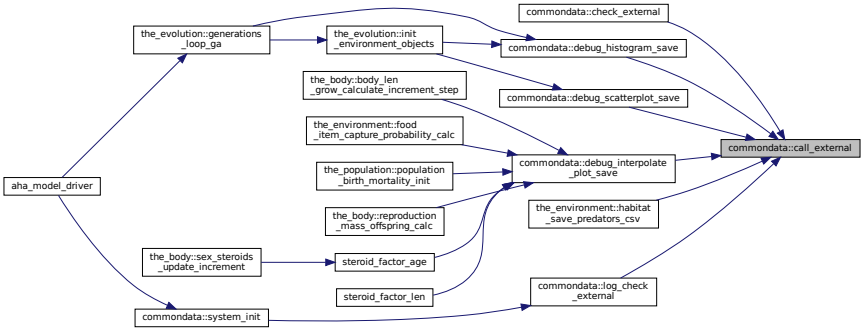
Warning

The intrinsic procedure `execute_command_line` is part of F2008 standard and may not be implemented yet on all platforms and compilers.

It is possible to get the command execution status if the logical `cmd_is_success` flag is provided.
Finally, log the command and its reported exit status if in the DEBUG mode.
Definition at line 7710 of file m_common.f90.
Here is the call graph for this function:



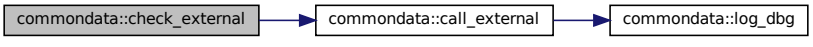
Here is the caller graph for this function:



8.1.3.57 check_external()

```
logical function commondata::check_external (
            character(len=*), intent(in) exec )
```

Check if an external procedure is executable and can be run.
Definition at line 7896 of file m_common.f90.
Here is the call graph for this function:



8.1.3.58 log_check_external()

```
subroutine commondata::log_check_external (
            character(len=*), intent(in) exec,
```

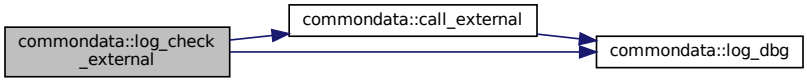
```
            logical, intent(in), optional debug_only,
            logical, intent(out), optional is_valid )
```

Check if an external procedure can be called and log the result.

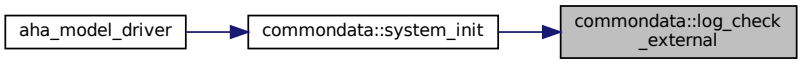
Parameters

in	*exec*	external executable name to call.
in	*debug_only*	flag indicating that checking is only done in the debug mode.
out	*is_valid*	returns if the external procedure is executable (TRUE) or not (FALSE).

Definition at line 7912 of file m_common.f90.
Here is the call graph for this function:



Here is the caller graph for this function:



8.1.3.59 debug_histogram_save()

```
subroutine commondata::debug_histogram_save (
            real(srp), dimension(:), intent(in) x_data,
            logical, intent(in), optional delete_csv,
            character(len=*), intent(in), optional csv_out_file,
            logical, intent(in), optional enable_non_debug )
```

Produce a **debug plot** of histogram using an external program `hthist` from HEDTOOLS tools.

Parameters

in	*x_data*	The data to be plotted.
in	*delete_csv*	Logical flag, if TRUE, csv file will be deleted after plot is done.
in	*csv_out_file*	Optional plot file name, if absent will be auto-generated.
in	*enable_non_debug*	Optional flag to enable plot even in NON- DEBUG mode. Normally, all plot outputs are disabled in non- DEBUG mode (`IS_DEBUG` is `FALSE`, see commondata::is_debug) because this may significantly slow down execution and produce lots of big PostScript files.

8.1.3.59.1 Implementation notes The histogram plot is actually produced using a separate program with the executable name set by the character parameter constant `commondata::exec_histogram` (that should be in the system path). The plotting program code is normally part of the HEDTOOLS modelling tools and is placed in

HEDTOOLS%\tools folder.

Note

Building plotting tools is done with "`make tools`" in the main HEDTOOLS file hierarchy. Building requires PGPLOT library and can be easily done on Linux; on Windows building can be more tricky. See code and output of `commondata::exec_histogram`.

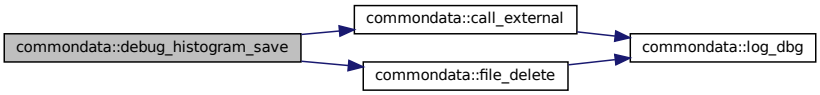
Debug plots are not essential, so are done by a separate program that can be absent on the runtime system. Plots are generated only if the protected parameter commondata::is_plotting is set to `TRUE`.

Normally, plots are generated only in the debug mode or if the `enable_non_debug` parameter is explicitly set to `TRUE`. This is because the code can easily generate huge number of plots. Also calling external commands has big calculation speed overhead and can exhaust OS-specific limits on child processes.
The plotting backend program obtains the input data for the scatterplot from a CSV file. Its name is normally provided on the command line. In this procedure, input data vector `x_data` is passed into the called plotting executable via a temporary CSV file. Its name can be automatically generated or provided explicitly as the `csv_out_file` dummy parameter. The CSV data file for plotting can also be saved. Therefore, the histogram could be easily regenerated from the data using an alternate program (e.g. high quality file prepared using a different program for publication).
First, the plot vector data are saved into the temporary CSV file.

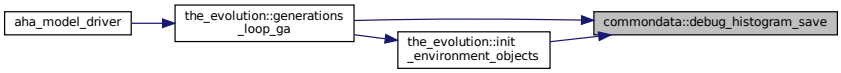
Note

Note that the data vector is saved *vertical* (`vertical=.TRUE.`).

Second, the external command to plot the data **histogram** is called using the `commondata::call_external()` wrapper procedure. We use only a one-dimensional vector of data and histogram is from the first column. No exit status or even the availability of `commondata::exec_scatterplot` is checked. However, the dummy parameter `is_background_task` controls whether the plotting program should be executed as a parallel background task (if `TRUE`) or the model executable should wait for the plotting program to terminate. The non-parallel (default) mode is safer because calling numerous child processes can exhaust the system-specific limit on child processes resulting in an uncontrollable crash. However, non-parallel mode is obviously much slower.
Definition at line 7965 of file m_common.f90.
Here is the call graph for this function:



Here is the caller graph for this function:



8.1.3.60 debug_scatterplot_save()

```
subroutine commondata::debug_scatterplot_save (
             real(srp), dimension(:), intent(in) x_data,
             real(srp), dimension(:), intent(in) y_data,
             logical, intent(in), optional delete_csv,
             character(len=*), intent(in), optional csv_out_file,
             logical, intent(in), optional enable_non_debug )
```

Produce a **debug plot** of 2-d scatterplot using an external program `htscatter` from HEDTOOLS tools.

Parameters

in	*x_data*	The data to be plotted.
in	*delete_csv*	Logical flag, if TRUE, csv file will be deleted after plot is done.
in	*csv_out_file*	Optional plot data file name, if absent will be auto-generated.
in	*enable_non_debug*	Optional flag, if TRUE, the plot output is saved even when *not* in the DEBUG mode. Normally, all plot outputs are disabled in non-DEBUG mode (commondata::is_debug is FALSE, see commondata::is_debug) because this may significantly slow down execution and produce lots of big PostScript files.

8.1.3.60.1 Implementation notes The scatterplot is actually produced with a a separate program with the executable name set by the character parameter constant commondata::exec_scatterplot (that should be in the system path). The scatterplot program code is normally part of the HEDTOOLS modelling tools and is placed in HEDTOOLS\tools folder.

Note

Building plotting tools is done with "make tools" in the main HEDTOOLS file hierarchy. Building requires PGPLOT library and can be easily done on Linux; on Windows building can be more tricky. See code and output of commondata::exec_scatterplot.

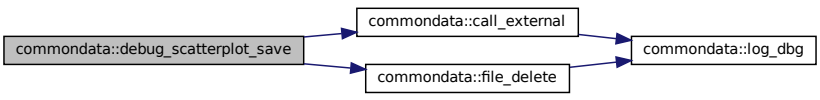
Debug plots are not essential, so are done by a separate program. They may be especially useful for controlling the quality of polynomial DDPINTERPOL interpolation, that has a tendency towards "wrapped" ends. Plots are generated only if the protected parameter commondata::is_plotting is set to TRUE.

Plots are normally produced only in the debug mode or if the enable_non_debug parameter is explicitly set to TRUE.
The plotting backend program obtains the input data for the scatterplot from a CSV file. Its name is normally provided on the command line. In this procedure, input data vectors x_data and y_data are passed into the called plotting executable via a temporary CSV file. Its name can be automatically generated or provided explicitly as the csv_out_file dummy parameter. The CSV data file for plotting can also be saved. Therefore, the scatterplot could be easily regenerated from the data using an alternate program (e.g. high quality file prepared using a different program for publication).
First, the vector data are saved into the temporary CSV file.

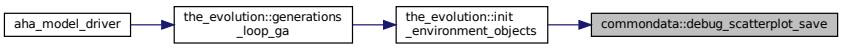
Note

Note that if X and Y vectors are different size, the bigger is used for the reshaped array and extra values are padded with MISSING (see commondata::missing).

Second, the external command to produce the plot is called using the commondata::call_external() wrapper procedure. No exit status or even the availability of commondata::exec_scatterplot is checked. However, the dummy parameter is_background_task controls whether the plotting program should be executed as a parallel background task (if TRUE) or the model executable should wait for the plotting program to terminate. The non-parallel (default) mode is safer because calling numerous child processes can exhaust the system-specific limit on child processes resulting in an uncontrollable crash. However, non-parallel mode is obviously much slower.
Definition at line 8095 of file m_common.f90.
Here is the call graph for this function:



Here is the caller graph for this function:



8.1.3.61 debug_interpolate_plot_save()

```
subroutine commondata::debug_interpolate_plot_save (
             real(srp), dimension(:), intent(in) grid_xx,
             real(srp), dimension(:), intent(in) grid_yy,
             real(srp), intent(in) ipol_value,
             character(*), intent(in) algstr,
             character(len=*), intent(in) output_file,
             logical, intent(in), optional enable_non_debug )
```

Produce a **debug plot** of the **interpolation data** using an external program `htinterp` from the HEDTOOLS tools.

Parameters

grid_xx	Interpolation grid arrays.
grid_yy	Interpolation grid arrays.
ipol_value	Interpolation value.
algstr	Algorithm string.
output_file	The file name for debug plot output (PostScript).
enable_non_debug	Optional flag, if TRUE, interpolation plot is saved even when *not* in the DEBUG mode. Normally, all plot outputs are disabled in non-DEBUG mode (`IS_DEBUG` is `FALSE`, see `commondata::is_debug`) because this may significantly slow down execution and produce lots of big PostScript files.

8.1.3.61.1 Implementation notes Plots are generated only if the protected parameter commondata::is_plotting is set to `TRUE`. So the first thing is to check if it is so.
By default, saving interpolation plots in non debug mode is disabled (.FALSE.)
Produce a **debug plot** of the interpolation data. The plot is done by a separate program with the executable name set by the character parameter `commondata::exec_interpolate` (that should be in the system path). This program is called is called using the `commondata::call_external()` wrapper procedure. The interpolation plot program code is normally part of the HEDTOOLS modelling tools and is placed in `HEDTOOLS\tools` folder.

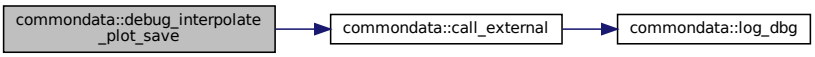
Note

Building plotting tools is done with "`make tools`" in the main HEDTOOLS file hierarchy. Building requires PGPLOT library and can be easily built on Linux; on Windows building can be more tricky. See code and output of `commondata::exec_interpolate`.

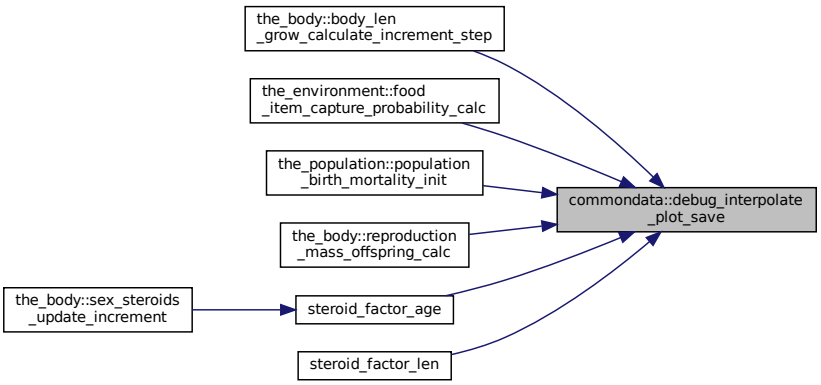
Debug plots are not essential, so are done by a separate program. They may be especially useful for controlling the quality of polynomial `DDPINTERPOL` interpolation, that has a tendency towards "wrapped" ends.

All the data for plotting are transferred into the plotting program via command line parameters. Each of the array or parameter should be in square brackets. No exit status or even the availability of `commondata::exec_interpolate` is checked. However, the dummy parameter `is_background_task` controls whether the plotting program should be executed as a parallel background task (if `TRUE`) or the model executable should wait for the plotting program to terminate. The non-parallel (default) mode is safer because calling numerous child processes can exhaust the system-specific limit on child processes resulting in an uncontrollable crash. However, non-parallel mode is obviously much slower.
Definition at line 8229 of file m_common.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.3.62 file_delete()

```
subroutine commondata::file_delete (
            character(len=*), intent(in) file_name,
            logical, intent(out), optional success )
```

Delete a file from the local file system using Fortran open status=delete or fast POSIX C call.

Note

The easiest way is to use `unlink` but it is non-standard (GNU extension).

HEDTOOLS now implement a few POSIX procedures for manipulation of the file system. Among them, `FS_UNLINK()` is used to delete a file (**not** directory) and `FS_REMOVE()` deletes a file or a directory. Example call:

```
call FS_UNLINK("obsolete_file", iostat)
```

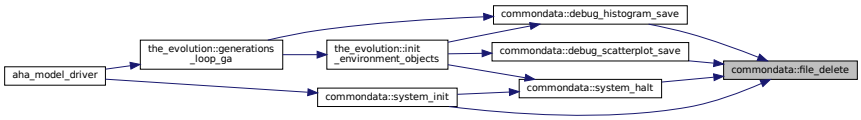
`iostat` is an optional integer error status of the operation.

Warning

POSIX-based `FS_UNLINK()` is now used in this procedure if the parameter `commondata::use_posix_fs_utils` is set to TRUE. See portability note on this parameter.

Definition at line 8322 of file m_common.f90.
Here is the call graph for this function:

Here is the caller graph for this function:



8.1.3.63 random_add_subtract()

```
real(srp) function, private commondata::random_add_subtract (
            real(srp), intent(in) x,
            real(srp), intent(in) y )  [private]
```

Random operator, adds or subtracts two values with equal probability, used in the random walk functions.

Parameters

in	*x*	x is the first real number
in	*y*	y is the second real number

Returns

x + *y* or *x* - *y* with equal 0.5 probability.

Definition at line 8384 of file m_common.f90.

8.1.3.64 system_init()

```
subroutine commondata::system_init
```

Initialises the system environment and sets basic parameters.

Note

This procedure also initialises the logger by a call of `commondata::logger_init`.

8.1.3.64.1 Check lock file First, the program checks if the lock file exists. If it does, this can mean that a simulation is already running in the working directory. Thus, the current process is terminated before any files are written to avoid mess.

Note

If the commondata::lock_file has not been deleted automatically for any reason but one is sure that no simulation is actually running (e.g. there was a crash before or the program was closed by Ctrl+C), just delete the commondata::lock_file and restart the program.

8.1.3.64.2 Random seed Then, the random seed is initialised by `RANDOM_SEED_INIT()` from `HEDTOOLS`. Note that it is platform-dependent and uses the system entropy pool on Unix systems and the date on Windows.

8.1.3.64.3 Execution control

8.1.3.64.3.1 The DEBUG mode The protected global variable `IS_DEBUG` (commondata::is_debug) sets up the debug mode of execution. The debug mode results in huge amount of output and logs that significantly slows down execution. Debug mode can be set using the environment variable `AHA_DEBUG=1`, `AHA_DEBUG=YES` or `AHA_DEBUG=TRUE`.
Debug mode can also be set by setting the runtime **command line** parameter to `DEBUG`, `DEBUG=1`, `DEBUG=YES` or `DEBUG=TRUE` to the model executable, e.g.

```
./MODEL.exe DEBUG
```

See The DEBUG mode for more details.

8.1.3.64.3.2 Logging in the DEBUG mode The procedure `CALL_DBG()` (see commondata::log_dbg()) is used to issue logger messages only in the DEBUG mode. Those messages, that should be issued in any mode, both DEBUG and non-DEBUG, `LOG_MSG()` from the HEDTOOLS should be used.
Examples: This code produces the logger message:

```
call LOG_MSG("Revision ID: " // SVN_Version // ".")
```

However, the below code produces logger output only in the debug mode:

```
call LOG_DBG(LTAG_INFO // "Agent is freezing.", PROCNAME, MODNAME)
```

Note the use of the logger tag commondata::ltag_info in the later example.

8.1.3.64.3.3 Compiler optimisation of the debug mode Setting the `IS_DEBUG` as a protected variable commondata::is_debug enables one to switch between the DEBUG and non-DEBUG mode of the program execution. However, it can have a performance penalty because all the calls to the debugging code remains in the model. The program then has to test numerous if conditions `if ( IS_DEBUG ) then...` which are likely to slow down execution. Setting commondata::is_debug as a fixed parameter (i.e. with the `parameter` attribute) would allow a highly optimising compiler to determine at the compile time that the numerous debugging instructions are never executed and remove them from the machine instructions that are then generated. However, declaring parameter won't allow to change the value of the `IS_DEBUG`. All such cases should be disabled. Within the code, such places are marked with the DEBUG_COMPILER_OPTIMISE tag. If `IS_DEBUG` is declared as a parameter, all these places should be disabled, e.g.

```
trim(debug_string)=="true"  ) then
! DEBUG_COMPILER_OPTIMISE: see 'Compiler optimisation of debug mode'
! IS_DEBUG =.TRUE.
is_screen_output = .true.
```

In effect, it would not be possible to switch between the debug modes on the fly.

8.1.3.64.3.4 Controlling the screen output The logger outputs normally go to the log file. But can also be shown on the terminal screen (standard output). This is controlled by the protected global variable `commondata::is_screen_output`.
The Screen mode can be reset independently of DEBUG mode using the shell environment variable `AHA_SCREEN=1`, `AHA_SCREEN=YES` or `AHA_SCREEN=TRUE`. For example, on Linux it is done

```
export AHA_SCREEN=1
```

on Windows command line:

```
set AHA_SCREEN=1
```

It is possible to change the output channel of the logger during the run time using the `LOG_CONFIGURE()` procedure. For example certain log message can selectively appear on the terminal standard output, and then the default behaviour controlled by the commondata::is_screen_output parameter restored:

```
call LOG_CONFIGURE("writeonstdout" , .TRUE.) ! Enable screen temporarily
call LOG_MSG( "*** START DEBUG BLOCK ***" )  ! Send message
call LOG_CONFIGURE("writeonstdout" , IS_SCREEN_OUTPUT) ! Enable default
```

For more details see the `LOGGER Module` documentation of the HEDTOOLS.

8.1.3.64.3.5 Generation of debug plots The debug plots can be generated. However, their number is normally just huge. Also, debug plots are produced by calling external programs which, if done too frequently, can exhaust the system-specific limit on the number of child processes. Production of debug plots is globally by the protected global variable `commondata::is_plotting`.
This can be reset independently of using the shell environment variable `AHA_DEBUG_PLOTS=1`, `AHA_DEBUG_PLOTS=YES` or `AHA_DEBUG_PLOTS=TRUE`. For example, on Linux it is done
```
export AHA_DEBUG_PLOTS=1
```
on Windows command line:
```
set AHA_DEBUG_PLOTS=1
```

8.1.3.64.3.6 Compression of big output data files The model can generate big data files for the agent population and habitat objects. These files can be automatically compressed using the external command defined by the commondata::cmd_zip_output parameter. This compression option is determined by the protected global variable `commondata::is_zip_outputs`.
This can be reset using the shell environment variable `AHA_ZIP_FILES=1`, `AHA_ZIP_FILES=YES` or `AHA_ZIP_FILES=TRUE`. For example, on Linux it is done
```
export AHA_ZIP_FILES=1
```
on Windows command line:
```
set AHA_ZIP_FILES=1
```

8.1.3.64.3.7 Checking external executables This procedure checks external procedures for existence and being executable. However, there seems to be a `bug in Intel Fortran` implementation of the intrinsic `execute_command_line` procedure: if the called program is not found in the `PATH`, the whole program crashes with "Segmentation fault".
```
forrtl: severe (174): SIGSEGV, segmentation fault occurred
```
A workaround is to avoid checking the externals in such a case. This can be done by setting the environment variable `AHA_CHECK_EXTERNALS` to `0`, `no` or `false`.

Note

This is not normally required if GNU gfortran is used for building. For example, on Linux it is done
```
export AHA_CHECK_EXTERNALS=NO
```

on Windows command line:
```
set AHA_CHECK_EXTERNALS=NO
```

8.1.3.64.4 Implementation details

- Get the time tag for the model in this format (YYYYMMDD) by call to the `commondata::tag_mmdd()` procedure. commondata::mmdd is a global public protected variable. It should be used for all date tagging.

Initialise the system logger by `commondata::logger_init`. Some system parameters, e.g. the platform type are determined in the `commondata::logger_init`.

8.1.3.64.4.1 Quick integrity checks

- Check and make sure IEEE float point errors are supported and not present at the start
- Check external executables that are called from the model code. These are basically not essential and mainly used only in the debug mode.
- If the commondata::is_zip_outputs is enabled (TRUE), a check is done if the external compression program (commondata::cmd_zip_output) can be called. It involves compressing a small test file and getting the exit code.

Warning

commondata::log_check_external() procedure is called in the synchronous mode and might hang the system if the child process hangs for any reason, e.g. if the compression utility is wrongly configured and waits input from the standard input.

If compression fails, automatic background compression is disabled. commondata::is_zip_outputs is set to FALSE.

- The results of these checks are reported in the logger.

If **any** of the external *plotting utilities* cannot be executed:

- commondata::exec_interpolate
- commondata::exec_scatterplot
- commondata::exec_histogram

plotting is disabled by setting commondata::is_plotting to FALSE.

- Check automatic allocation of arrays on intrinsic assignment. The code here and there depends on this recent feature of Fortran. If the test fails, manual allocation of such arrays with `allocate` should be recoded. The test is tagged as `TEST_AUTOALLOC`.
- Check automatic determination of the size in parameter parameter arrays of the dimension statement style: `dimension(*)`. All fairly modern Fortran compilers should support this feature.

Note

If the compiler does not support `dimension(*)` it would probably just fail to compile this code. So, it won't run up to this point.

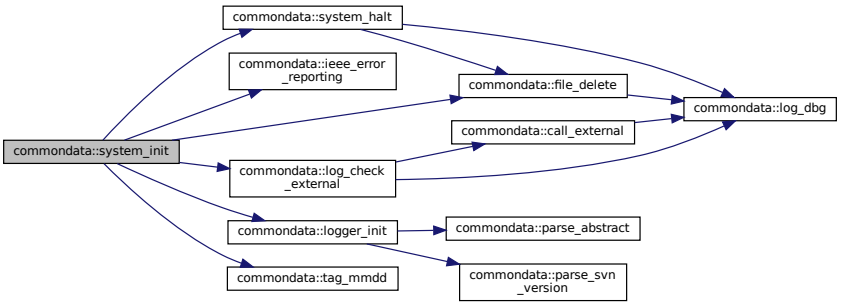
- If fatal tests (`TEST_AUTOALLOC`, `TEST_AUTO_PARAM_ARRAYS`) fail, commondata::system_halt() is called

Finally, a lock file is created. This file keeps opened during the whole simulation and is closed and deleted at the end. Thus, its primary use is to signal that the simulation is still going. See commondata::lock_file and commondata::lock_file_unit and The lock file for details.

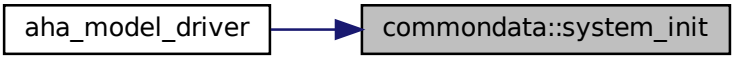
Note

Lock file operation uses native Fortran intrinsic `open` statement rather than any higher level procedures like file_io. This is because the file is for signalling only (intended to be empty) and nothing is actually written into it.

Note that if the lock file cannot be written, the simulation is not halted automatically, but an error is issued to the logger because such error signals severe problems with disk access (e.g. read only mode, no space left etc.).
Definition at line 8404 of file m_common.f90.
Here is the call graph for this function:



Here is the caller graph for this function:



8.1.3.65 system_halt()

```
subroutine commondata::system_halt (
            logical, intent(in), optional is_error,
            character (len=*), intent(in), optional message,
            logical, intent(in), optional ignore_lockfile )
```

Halt execution of the system with a specific message and exit code. The exit code is normally passed to the operating system. However, this behaviour is implementation dependent and can be unexpected on specific the platform(s) and the compiler(s).

- Checking the exit code in bash can be done with "`$?`" variable.
- Checking exit code on Windows is done using "`%ERRORLEVEL%`" variable e.g.
  ```
  if %ERRORLEVEL% EQU 0 echo normal termination
  ```

Note

Using specific exit code could allow to place the simulation model into an automated batch job more easily. For example, several simulations with different parameters could be processed. In such a case it would be good to know if the program failed or not. An example termination call due to an error:
```
call system_halt(is_error=.TRUE., message=ERROR_NO_AUTOALLOC )
```

Parameters

in	*is_error*	is_error Optional flag that signals that the program is terminating due to error. The default is normal error-free termination with zero exit code.
in	*message*	message Optional message that is passed the the logger immediately before the program is terminated.
in	*ignore_lockfile*	ignore_lockfile is an optional flag to ignore closing and deleting the lock file. The default value is FALSE. If it is TRUE, the lock file is not touched. This is primarily necessary to halt the execution because the program discovers the lock file on startup which may indicate another simulation is currently running here. In such a case, deleting the lock file would interfere with this pre-running simulation.

8.1.3.65.1 Implementation notes

There are two possible exit code:

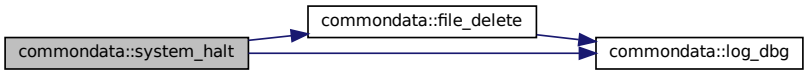
- **EXIT_CODE_DEF** is the default exit code that is returned to the operating system. The default value for error-free termination is zero.

EXIT_CODE_ERROR is the fixed exit code that is returned to the operating system in case of error.
The lock file commondata::lock_file is closed and then deleted (commondata::file_delete()). See commondata::lock_file and commondata::lock_file_unit and The lock file for details.

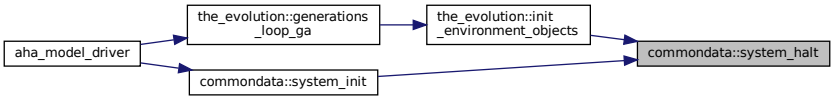
Note

Note that setting `ignore_lockfile` parameter to TRUE disables checking and deleting the lock file.

The logger is now issues the final messages and shuts down.
The final message goes to the standard error device.
And the program terminates with specific exit code.
Definition at line 8867 of file m_common.f90.
Here is the call graph for this function:



Here is the caller graph for this function:



8.1.3.66 logger_init()

`subroutine, private commondata::logger_init [private]`
logger_init Initialise the system and the system logger.
`logger_init` is called only once at `commondata::system_init()` to set up the basic parameters and the logging facility. For example, it sets the log file name, if timestamps should be produced, if we need screen output, log delimiter characters and any other similar parameters. They can be changed later if needed. `logger_init` also writes some short initial information to the log file like model name etc.

8.1.3.66.1 Implementation details

8.1.3.66.1.1 Notable variables **logfile** character variable defines the file name for the main log output. Such a log file name is normally assembled from pieces, such as model name, date and time tag etc.
run_on_hostname is an character string variable that keeps the hostname of the system the model is running on.

Warning

Hostname is determined using the `hostnm` function that is non-standard and not portable. However, it is supported by GNU and Intel Fortran compilers. If not supported by the currently used compiler, call to this function should be disabled in this procedure.

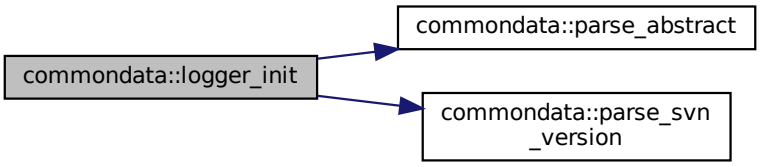
8.1.3.66.1.2 The procedure We first set parameters of the log, switch timestamps, screen output when DEBUG=TRUE, set delimiters and file optionally unit.
Delimiters here have the length 60 character.
We then initialise the log and set the log file name. The second dummy parameter set to FALSE defines log file overwrite (don't append to the old log file).
We also print some some initial info to the log, like the name of the model.
The Model Abstract is obtained from the abstract file commondata::model_abstract_file and logged.
Then detect the runtime platform and set integer platform ID that shows if we are running on Windows or Unix.

Determine the compiler version and the compiler parameters using the F2008 `compiler_version()` and `compiler_options()` inquiry functions. Because this functionality may not have been implemented in many current compilers (e.g. Intel Fortran 17), it is disabled here.
Determine the hostname (computer name) the program is running on. Then log hostname if determined successfully.

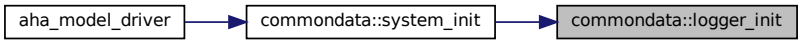
Warning

Hostname is determined using the `hostnm` function that is non-standard and not portable. However, it is supported by GNU and Intel Fortran compilers. If not supported by the currently used compiler, call to this function should be disabled in this procedure. With Intel Fortran compiler, using this functionality requires declaring. use IFPORT, only : hostnm Therefore, determining hostname is disabled so far.

Print the logger parameters in to the log itself.
Print some basic constants, e.g. commondata::srp, commondata::hrp etc.
Print the main parameters of the model, population size etc.
Print also some parameters of the Genetic Algorithm.
Finally, issue a horizontal line delimiter into the log. This finishes initialising the logger.
Definition at line 8977 of file m_common.f90.
Here is the call graph for this function:



Here is the caller graph for this function:



8.1.3.67 log_dbg()

```
subroutine commondata::log_dbg (
            character (len=*), intent(in) message_string,
            character (len=*), intent(in), optional procname,
            character (len=*), intent(in), optional modname )
```

`LOG_DBG`: debug message to the log. The message goes to the logger only when running in the DEBUG mode.

Parameters

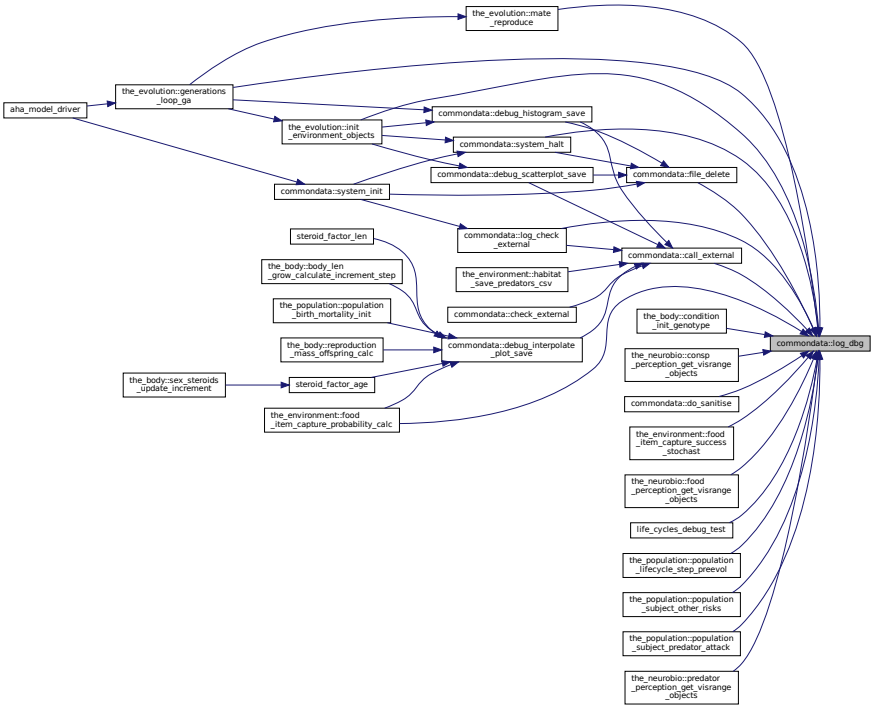
in	*message_string*	String text for the log message
in	*procname*	Optional procedre name for debug messages
in	*modname*	Optional module name for debug messages

This subroutine is a wrapper to LOG_MSG() from HEDTOOLS for writing debug messages by the module LOGGER. The debug message message defined by the message_string parameter is issued only when the model runs in the debug mode, i.e. if IS_DEBUG=.TRUE.

Note

Standard LOG_MSG() procedure should be used for all logger messages that are produced in the normal non debug mode. How the DEBUG mode is controlled is described in the commondata::system_init() reference.

We first generate the message prefix = MODNAME PROCNAME if called with these parameters, so the location of the code in which the message has been issued is precisely known.
And then we issue the message to the log as usual.
Definition at line 9170 of file m_common.f90.
Here is the caller graph for this function:



8.1.3.68 log_ieee()

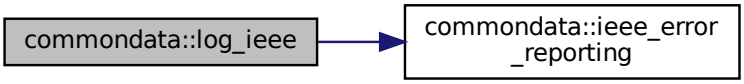
```
subroutine commondata::log_ieee (
            character(len=*), intent(in), optional ttag,
            logical, intent(in), optional always_log,
            logical, intent(in), optional reset_flags )
```

LOG_IEEE: Check and log IEEE signalling flags. Logging normally occurs only if any nonzero output from ieee_error_reporting() is found.

Parameters

in	*ttag*	Optional text tag that is passed to logger to identify the specific algorithm place where the IEEE condition is checked/flagged.
in	*always_log*	Optional flag to request logger output even if IEEE signalling is NOT reported. Normally defaults to FALSE, but to TRUE if is_debug is TRUE.
in	*reset_flags*	Optional flag to reset all IEEE conditions after checking and logging. The default value is TRUE, i.e. reset all flags to zero

Definition at line 9214 of file m_common.f90.
Here is the call graph for this function:



8.1.3.69 parse_svn_version()

`character(len=:) function, allocatable, private commondata::parse_svn_version [private]`
Parse and cut revision **number** in form of string from the whole SVN revision string. SVN revision number can therefore be included into the model outputs and output file names. This is convenient because the model version is identified by a single SVN revision number.

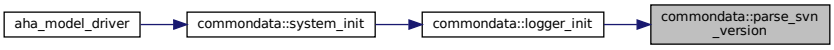
Returns

revision number from Subversion.

Warning

`STRINGS` module uses unsafe coding prone to bugs, e.g. does not clearly state dummy parameters intent and doesn't work correctly with `parameter`s.

8.1.3.69.1 Implementation notes Subversion has a useful feature: various keywords can be inserted and automatically updated in the source code under revision control, e.g. revision number, date, user etc. The character string parameter constant `commondata::svn_version_string` keeps the Subversion revision tag. This subroutine parses the tag striping all other characters out.
Definition at line 9273 of file m_common.f90.
Here is the caller graph for this function:



8.1.3.70 parse_abstract()

```
character(len=long_label_length) function, dimension(:), allocatable commondata::parse_↩
abstract (
            character(len=*), intent(in), optional file_name )
```
Get and parse the model abstract. Model abstract is a short descriptive text that can span several lines and is kept in a separate file that is defined by the commondata::model_abstract_file.
The separate Model Abstract file is useful because it can integrate dynamic information, such as the latest version control log(s) via Subversion or Mercurial hooks mechanism.

Parameters

in	*file_name*	file_name optional name of the abstract file. If this parameter is absent, commondata::model_abstract_file is used.

Returns

An allocatable character string array that contains all the lines from the Model Abstract file.

Try to open the abstract file and count its lines with the function CSV_FILE_LINES_COUNT() from HEDTOOLS.

- If the function return status is 'error' or the number of lines is less than 1, the abstract file is invalid and the abstract array is automatically allocated to one and assigned the model description string from commondata::model_descr.
- Otherwise,
  - allocate the output abstract array to the number of lines in the abstract file and initialise it to empty strings
- get the free file unit for reading...
- open the abstract file for reading
- And read all the contents of the file into the output abstract array.
- Finally, close the file.

Definition at line 9304 of file m_common.f90.
Here is the caller graph for this function:

8.1.3.71 tag_mmdd()

`character(len=:) function, allocatable, private commondata::tag_mmdd [private]`
Date (YYYYMMDD) tag for file names and logs.

Warning

Note that this procedure should be called only once during the system initialisation. Then a fixed value is given to the global variable commondata::mmdd: this global variable should be used for all tags. This procedure has *private* accessibility status and is not available outside of commondata.

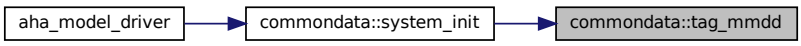
Returns

Return values

MMDD	Function returns an 8-character string for YYYYMMDD

Definition at line 9387 of file m_common.f90.

Here is the caller graph for this function:



8.1.4 Variable Documentation

8.1.4.1 s_prec_32

```
integer, parameter, public commondata::s_prec_32 = selected_real_kind( 6, 37)
```

Standard precision for real data type. We first define 32, 64 and 128 bit real kinds.

Warning

HEDTOOLS cannot accept precision higher than kind 8 so far. So 128 bit reals are for example only here. Have to implement higher precision HEDTOOLS routines if they are really used.

Definition at line 1534 of file m_common.f90.

8.1.4.2 d_prec_64

```
integer, parameter, public commondata::d_prec_64 = selected_real_kind(15, 307)
```

Definition at line 1535 of file m_common.f90.

8.1.4.3 q_prec_128

```
integer, parameter, public commondata::q_prec_128 = selected_real_kind(33, 4931)
```

Definition at line 1536 of file m_common.f90.

8.1.4.4 srp

```
integer, parameter, public commondata::srp = S_PREC_32
```

Definition of the **standard** real type precision (*SRP*).
SRP is defined as the standard precision that should normally be used for all real variables and constants. **SRP** stands for **Standard Real Precision** (Naming note: const name should be short to not produce too long real definitions, e.g. `real(SRP) :: alpha`).

Warning

All float (and other) constants should ideally be defined in the definition section of COMMONDATA or another module, **not** in the code. It is for easier maintainability and precision control.

All standard real variables should be defined as: `real(SRP) :: real_var`. **Literal constants** should normally add `_SRP`: `1.234_SRP` (although it is less crucial).

Definition at line 1551 of file m_common.f90.

8.1.4.5 hrp

```
integer, parameter, public commondata::hrp = Q_PREC_128
```

Definition of the **high** real precision (**HRP**). This real type kind is used in pieces where a higher level of FPU precision is required, e.g. to avoid overflow/underflow and similar errors.
Definition at line 1556 of file m_common.f90.

8.1.4.6 long

```
integer, parameter, public commondata::long = selected_int_kind(16)
```

In some (perhaps quite rare) cases of exponentiation we may also need huge integers, those in 64 bit would probably be enough. So whenever we need such a big integer, declare it as:

```
integer(LONG) :: bignum
```

Warning

HEDTOOLS **do not** currently work with these `LONG` kind integers. So they are only for "internal"-calculation use. Alternatively, use the intrinsic function `int` to convert to the default integer type inline before use, e.g.: `TOSTR(int(max_permutations))`.

Definition at line 1569 of file m_common.f90.

8.1.4.7 modname

```
character(len=*), parameter, private commondata::modname = "(COMMONDATA)"  [private]
```

MODNAME always refers to the name of the current module for use by the LOGGER function LOG_DBG. Note that in the debug mode (if IS_DEBUG=TRUE) LOGGER should normally produce additional messages that are helpful for debugging and locating possible sources of errors. MODNAME is declared private and is not accessible outside of this module. Each procedure should also have a similar private constant `commondata::procname`.

Note

`MODNAME` must have the same case as the module name itself and must be enclosed in parentheses, e.g. `"(THE_MODULE)"`.

Definition at line 1591 of file m_common.f90.

8.1.4.8 procname

```
character(len=*), parameter, private commondata::procname = ""  [private]
```

PROCNAME is the procedure name for logging and debugging (with `commondata::modname`).

Note

`PROCNAME` must have the same case as the subroutine itself and must be enclosed in parentheses, e.g. `"(function_or_subroutine_name)"`, so that it is easier to find in the code and easy to search by regex in the long output logs (use parentheses to search). Here is a template to insert into the code (procedure name is to be filled in the parentheses):

```
! PROCNAME is the procedure name for logging and debugging
character(len=*), parameter :: PROCNAME = "()"
```

Definition at line 1605 of file m_common.f90.

8.1.4.9 svn_version_string

```
character(len=*), parameter, public commondata::svn_version_string = "$Revision:  9552 $"
```

Subversion or *Mercurial* revision number (or ID) of the model code.

Warning

The revision string is **updated automatically** at svn commmit (or hg commit). It is also not fully portable and may **not** auto update if a different version control system is used. Note that *Mercurial* has a **keyword** extension that works similar to Subversion and should auto-update the keywords.

Definition at line 1613 of file m_common.f90.

8.1.4.10 svn_version

`character(len=:), allocatable, public, protected commondata::svn_version`
Subversion or *Mercurial* revision number that is parsed by `commondata::parse_svn_version()`. It is shorter than `commondata::svn_version_string` and does not contain blanks. Therefore, it can be used for building output file names.

Note

Note that the SVN parse function is called at initialising the log `LOGGER_INIT`, so `SVN_Version` is initialised.

Warning

`SVN_Version` is a string, version ID in *Subversion* is numeric, but in other version control systems (hg or git) it can be an arbitrary non-numeric hash string.

Note

Because it has allocatable attribute, its actual length is obtained automatically and no `trim(SVN_Version)` is necessary.

Definition at line 1627 of file m_common.f90.

8.1.4.11 true

`logical, parameter, public commondata::true =.TRUE.`
Safety parameter avoid errors in logical values, so we can now refer to standard Fortran `.TRUE.` and `.FALSE.` as `YES` and `NO` or `TRUE` and `FALSE`
Definition at line 1632 of file m_common.f90.

8.1.4.12 false

`logical, parameter, public commondata::false =.FALSE.`
Definition at line 1632 of file m_common.f90.

8.1.4.13 yes

`logical, parameter, public commondata::yes =.TRUE.`
Definition at line 1633 of file m_common.f90.

8.1.4.14 no

`logical, parameter, public commondata::no =.FALSE.`
Definition at line 1633 of file m_common.f90.

8.1.4.15 zero

`real(srp), parameter, public commondata::zero = epsilon(0.0_SRP)`
Some parameters should never be zero or below. In such cases they could be set to some smallest distinguishable non-zero value. Here set as the Fortran intrinsic `epsilon` function, a value that is almost negligible compared to one, i.e. the smallest `real` number *E* such that $1 + E > 1$. In some cases it is also reasonable to set the tolerance limit to this parameter (see Float point computations).

Note

The value of this parameter computed on a x86_64 Linux platform is `1.19209290E-07`.

Definition at line 1644 of file m_common.f90.

8.1.4.16 tiny_srp

`real(srp), parameter, public commondata::tiny_srp =tiny(1.0_SRP)`
The smallest positive number in the commondata::srp standard real model.

Note

This parameter is used for definition of numerical tolerance. See Float point computations.

The value of this parameter computed on a x86_64 Linux platform is `1.17549435E-38`.

Definition at line 1651 of file m_common.f90.

8.1.4.17 tiny_hrp

`real(hrp), parameter, public commondata::tiny_hrp =tiny(1.0_HRP)`
The smallest positive number in the commondata::hrp high precision real model. See Float point computations.

Note

This parameter is used for definition of numerical tolerance.

Definition at line 1656 of file m_common.f90.

8.1.4.18 lo_valid_sanitised

`real(srp), parameter, public commondata::lo_valid_sanitised = TINY_SRP * 10.0_SRP`
Lower bound for do_sanitise() procedure. This is the lowest value that considered valid.
Definition at line 1660 of file m_common.f90.

8.1.4.19 hi_valid_sanitised

`real(srp), parameter, public commondata::hi_valid_sanitised = huge(1.0_SRP)/100.0_SRP`
Higher bound for do_sanitise() procedure. This is the highest value that considered valid.
Definition at line 1664 of file m_common.f90.

8.1.4.20 tolerance_low_def_srp

`real(srp), parameter, public commondata::tolerance_low_def_srp = TINY_SRP * 5.0_SRP`
Default value of *low* tolerance (*high precision*). This is the *standard* commondata::srp precision. See Float point computations.

Note

The value of this parameter computed on a x86_64 Linux platform is `5.87747175E-38`.

Definition at line 1671 of file m_common.f90.

8.1.4.21 tolerance_low_def_hrp

`real(hrp), parameter, public commondata::tolerance_low_def_hrp = TINY_HRP * 5.0_HRP`
Default value of *low* tolerance (*high precision*). This is the *high* commondata::hrp precision. See Float point computations.
Definition at line 1676 of file m_common.f90.

8.1.4.22 tolerance_high_def_srp

`real(srp), parameter, public commondata::tolerance_high_def_srp = ZERO * 1000.0_SRP`
Default value of *high* tolerance (*low precision*). This is the *standard* commondata::srp precision real. See Float point computations.

Note

The value of this parameter computed on a x86_64 Linux platform is `1.19209290E-04`.

Definition at line 1683 of file m_common.f90.

8.1.4.23 tolerance_high_def_hrp

`real(hrp), parameter, public commondata::tolerance_high_def_hrp = epsilon(0.0_HRP) * 1000.0_↩ HRP`
Default value of *high* tolerance (*low precision*). This is the *high* commondata::hrp precision real. See Float point computations.

Note

The value of this parameter computed on a x86_64 Linux platform is `1.92592994438723585305597794258492732E-0`

Definition at line 1690 of file m_common.f90.

8.1.4.24 missing

`real(srp), parameter, public commondata::missing = -9999.0_SRP`
Numerical code for *missing* and *invalid* **real type** values.

Note

We deliberately set an unisually *big negative value* for `MISSING` because it will reveal bugs by clearly strange/invalid negative results that will propagate in calculations.

Warning

It is **safe** to assign integer `UNKNOWN` constant to a **real** type variable, see the next definition.

Definition at line 1699 of file m_common.f90.

8.1.4.25 invalid

`real(srp), parameter, public commondata::invalid = -9999.0_SRP`
Definition at line 1699 of file m_common.f90.

8.1.4.26 unknown

`integer, parameter, public commondata::unknown = -9999`
Numerical code for invalid or *missing* **integer** counts.

Note

It is safe to assign integer `UNKNOWN` to a **real** type variable, e.g. `real(SRP) :: value=UNKNOWN`.

Definition at line 1704 of file m_common.f90.

8.1.4.27 pi

`real(srp), parameter, public commondata::pi =4.0_SRP*atan(1.0_SRP)`
The **PI** number.
Pi number $\pi = 4.0 \cdot tg(1.0)$ `[4.*atan(1.0)]`, numerically equal to (64 bit real) `PI=3.14159265358979323846264338327_Q_PREC_64`.
Definition at line 1710 of file m_common.f90.

8.1.4.28 csv

`character(len=*), parameter, public commondata::csv =".csv"`
Standard data file extension for data output is now .csv.
Definition at line 1713 of file m_common.f90.

8.1.4.29 ps

`character(len=*), parameter, public commondata::ps =".ps"`
Standard file extension for debug and other PostScript plots.
Definition at line 1716 of file m_common.f90.

8.1.4.30 filename_length

`integer, parameter, public commondata::filename_length = 255`
Set the standard length of the file name, are 255 characters enough?

Warning

Do not forget to use `trim()` function to delete trailing spaces from the file name if it is declared as a fixed-length character string.

Definition at line 1722 of file m_common.f90.

8.1.4.31 use_posix_fs_utils

`logical, parameter, public commondata::use_posix_fs_utils = .TRUE.`
Logical flag for setting if POSIX direct filesystem procedures are used. These utilities are implemented in HED-TOOLS for standard POSIX C call via the Fortran interface. They should work safer, better and faster than indirect procedure wrappers (e.g. calling `system()`) but are not fully portable and might not work as expected on all systems and compilers.
Definition at line 1730 of file m_common.f90.

8.1.4.32 label_length

`integer, parameter, public commondata::label_length = 14`
The length of standard character string labels. We use labels for various objects, like alleles, perceptual and neural components / bundles etc. For simplicity, they all have the same length. It should be big enough to fit the longest whole label.
Definition at line 1736 of file m_common.f90.

8.1.4.33 long_label_length

```
integer, parameter, public commondata::long_label_length = 128
```
The length of long labels.
Definition at line 1739 of file m_common.f90.

8.1.4.34 label_cst

```
integer, parameter, public commondata::label_cst = 97
```
This parameter defines the range of characters that is used for generating random labels, 97:122 corresponds to lowercase Latin letters.
Definition at line 1743 of file m_common.f90.

8.1.4.35 label_cen

```
integer, parameter, public commondata::label_cen = 122
```
Definition at line 1743 of file m_common.f90.

8.1.4.36 lock_file

```
character(len=*), parameter commondata::lock_file = "lock_simulation_running.lock"
```
The name of the lock file. The lock file is created at the start of the simulation and is deleted at the end of the simulation. It can be used to signal that simulation is still ongoing to external utilities and scripts. See The lock file.

Note

> Lock file operation uses native Fortran intrinsic `open` statement rather than any higher level procedures like file_io. This is because the file is for signalling only (intended to be empty) and nothing is actually written into it.

Definition at line 1754 of file m_common.f90.

8.1.4.37 lock_file_unit

```
integer, public, protected commondata::lock_file_unit
```
This is the unit number that identifies the lock file. The lock file is created at the start of the simulation and is deleted at the end of the simulation. It can be used to signal that simulation is still ongoing to external utilities and scripts. See The lock file.

Note

> Lock file operation uses native Fortran intrinsic `open` statement rather than any higher level procedures like file_io. This is because the file is for signalling only (intended to be empty) and nothing is actually written into it.

Definition at line 1766 of file m_common.f90.

8.1.4.38 stop_file

```
character(len=*), parameter commondata::stop_file = "stop_simulation_running.lock"
```
The name of the stop file. The stop file is checked before each new generation of the Genetic Algorithm. If this file is found, simulation does not go to the next generation and just stops. See The stop file.

Note

Stop file operation uses native Fortran intrinsic `open` statement rather than any higher level procedures like file_io. This is because the file is for signalling only (intended to be empty) and nothing is actually read or written.

Definition at line 1777 of file m_common.f90.

8.1.4.39 platform_windows

```
integer, parameter, public commondata::platform_windows = 100
```

Runtime platform ID constants. Use these constants for determining the current runtime platform, e.g. `Platform_Running = PLATFORM_WINDOWS`. See `commondata::platform_running`.
Definition at line 1782 of file m_common.f90.

8.1.4.40 platform_unix

```
integer, parameter, public commondata::platform_unix = 111
```

Definition at line 1783 of file m_common.f90.

8.1.4.41 platform_running

```
integer, public commondata::platform_running
```

Global variable that shows what is the current platform. Should use the above platform constants, e.g. `Platform_Running = PLATFORM_WINDOWS`. See `commondata::platform_windows` and `commondata::platform_unix`.
Definition at line 1788 of file m_common.f90.

8.1.4.42 exec_interpolate

```
character(len=*), parameter, public commondata::exec_interpolate = "htintrpl.exe"
```

There are a few **external programs** which are called from the model code. The name of the **interpolation** program (htintrpl.f90 from HEDTOOLS) executable.
Definition at line 1793 of file m_common.f90.

8.1.4.43 exec_scatterplot

```
character(len=*), parameter, public commondata::exec_scatterplot = "htscatter.exe"
```

The name of the **scatterplot** program (htscatter.f90 from HEDTOOLS) executable.
Definition at line 1797 of file m_common.f90.

8.1.4.44 exec_histogram

```
character(len=*), parameter, public commondata::exec_histogram = "hthist.exe"
```

The name of the **histogram** program (hthist.f90 from HEDTOOLS) executable.
Definition at line 1801 of file m_common.f90.

8.1.4.45 ltag_major

```
character(len=*), parameter, public commondata::ltag_major = "IMPORTANT: "
```

Tag prefixes for the logger system. The log may use tags for some common information pieces, so they are easily found within. The tags are normally set the prefix for the log: 017-01-31 13:33:22 INFO: Saving histogram, data: debug_hist.csv Some common tags are: STAGE STAGE: 2017-01-31 16:03:15 INFO: Generation 7 took

448.3279s. INFO INFO: some information TIMER TIMER: Calculating distances took 0.001 s Tag meaning:

- **MAJOR** major stages of the simulation, e.g. next generation;
- **STAGE** minor stages of the simulation, e.g. time step change;
- **INFO** some information about model running;
- **WARNING** warnings on possible issues and minor errors;
- **ERROR** error reports that do not normally halt running;
- **CRITICAL** critical errors that would stop execution;
- **TIMER** reports from the timers and stopwatches.

Definition at line 1819 of file m_common.f90.

8.1.4.46 ltag_stage

```
character(len=*), parameter, public commondata::ltag_stage = "STAGE: "
```
Definition at line 1820 of file m_common.f90.

8.1.4.47 ltag_info

```
character(len=*), parameter, public commondata::ltag_info = "INFO: "
```
Definition at line 1821 of file m_common.f90.

8.1.4.48 ltag_warn

```
character(len=*), parameter, public commondata::ltag_warn = "WARNING: "
```
Definition at line 1822 of file m_common.f90.

8.1.4.49 ltag_error

```
character(len=*), parameter, public commondata::ltag_error = "ERROR: "
```
Definition at line 1823 of file m_common.f90.

8.1.4.50 ltag_crit

```
character(len=*), parameter, public commondata::ltag_crit = "CRITICAL: "
```
Definition at line 1824 of file m_common.f90.

8.1.4.51 ltag_timer

```
character(len=*), parameter, public commondata::ltag_timer = "TIMER: "
```
Definition at line 1825 of file m_common.f90.

8.1.4.52 ltag_stats

```
character(len=*), parameter, public commondata::ltag_stats = "STATS: "
```
Definition at line 1826 of file m_common.f90.

8.1.4.53 error_no_autoalloc

```
character(len=*), parameter, public commondata::error_no_autoalloc = "No automatic array allocation"
```

Error message for **"no automatic intrinsic array allocation"**. Fortran compilers support automatic allocation of arrays on intrinsic assignment. This feature should work by default in GNU gfortran v.4.6 and Intel ifort v.17.0.1. Automatic allocation allows to avoid a possible bug when the number of array elements in the `allocate` statement is not updated when the components of the array are updated in the array constructor.
Definition at line 1841 of file m_common.f90.

8.1.4.54 error_auto_param_arrays

```
character(len=*), parameter, public commondata::error_auto_param_arrays = "No automatic size
in parameter arrays"
```

Error message for **"no automatic determination of the size in parameter"** arrays in the style:
`real(SRP), parameter, dimension(*) :: ARRAY=[ 1.0, 2.0, 3.0, 4.0 ]`
Definition at line 1849 of file m_common.f90.

8.1.4.55 error_allocation_fail

```
character(len=*), parameter, public commondata::error_allocation_fail = "Cannot allocate array
or object"
```

Error message **"Cannot allocate array or object"** is issued if an array or an object is checked and turns out to be not allocated while it must be.
Definition at line 1855 of file m_common.f90.

8.1.4.56 error_lock_preexists

```
character(len=*), parameter, public commondata::error_lock_preexists = "Lock file '" // LOCK_↩
FILE // "' exists.  Is another simulation running?"
```

Definition at line 1858 of file m_common.f90.

8.1.4.57 model_name

```
character (len=*), parameter, public commondata::model_name = "HEDG2_04"
```

Model name for tags, file names etc. Must be very short. See Model descriptors.
Definition at line 1938 of file m_common.f90.

8.1.4.58 model_descr

```
character (len=*), parameter, public commondata::model_descr = "AHA, single fear, body size
non-genetic."
```

Model description - a fixed descriptive text, used in text outputs etc. See Model descriptors.
Definition at line 1942 of file m_common.f90.

8.1.4.59 model_abstract_file

```
character (len=*), parameter, private commondata::model_abstract_file = "abstract.txt"  [private]
```

The name of the file that contains the Model abstract, a short description that can span several lines of text ans is kept in a separate file. The file is read, if it exists, and its contents is logged at the start the simulation. The separate Model Abstract file is useful because it can integrate dynamic information, such as the latest version control log(s) via Subversion or Mercurial hooks mechanism. See Model descriptors.
Definition at line 1953 of file m_common.f90.

8.1.4.60 is_debug

`logical, public, protected commondata::is_debug =.FALSE.`
Sets the model in the debug mode if TRUE. The Debug mode generates huge additional outputs and logs. Also, the logs by default go to the screen (standard output). See `commondata::system_init()` for details.

Warning

This is a protected variable not fixed parameter. Can be set at start from the command line parameter or environment variable.

Note

Debug mode can be set in three ways (see `system_init` subroutine):

1. by setting this variable at the initialisation here to TRUE (although this is **not-normal** as requires recompile);
2. by setting the shell environment variable `AHA_DEBUG=1`, `AHA_DEBUG=TRUE` or `AHA_DEBUG=YES`;
3. by setting the command line parameter `DEBUG` when calling this executable program.

Warning

`IS_DEBUG` can also be declared as a `parameter`, see Compiler optimisation of debug mode in commondata::system_init() for details.

Note

`IS_DEBUG` can also be declared as a normal global variable, not "protected" (by removing the `protected` attribute) In such case it could be changed everywhere in the program. A potential benefit is that only a section of the program, e.g. a single function, can then produce extended debugging output. However, this would also make compiler optimisation more difficult and reduce performance. See commondata::system_init() for more discussion.

Remarks

A short discussion on the protected attribute for module variable is at `comp.lang.fortran`.

Definition at line 1981 of file m_common.f90.

8.1.4.61 is_plotting

`logical, public, protected commondata::is_plotting =.TRUE.`
This parameter controls if the debug plots are produced. They can be huge number that takes lots of space. Also, debug plots are called as separate processes that can run at the background and easily exceed the system-specific limit on child processes (if run in asynchronous mode). Generation of debug plots can be controlled by the environment variable `AHA_DEBUG_PLOTS`: if it is set to TRUE, 1, or YES, debug plots are enabled. See `commondata::system_init()` for details.
Definition at line 1992 of file m_common.f90.

8.1.4.62 is_screen_output

`logical, public, protected commondata::is_screen_output =.FALSE.`
Sets the model in screen output mode. If TRUE, the logger output goes to the screen (standard output device). Can be manipulated using the environment variable `AHA_SCREEN`. If `AHA_SCREEN` is set to TRUE or 1 or yes, logger screen output is enabled. See `commondata::system_init()` for details.
Definition at line 1999 of file m_common.f90.

8.1.4.63 is_zip_outputs

`logical, public, protected commondata::is_zip_outputs =.FALSE.`
This parameter enables or disables post-processing compression of the data: if TRUE, the data are compressed using the command defined by the commondata::cmd_zip_output string parameter. Note that not all data files are compressed, only potentially big ones are (e.g. agent population data and habitat data).

Note

Note that each data file is compressed individually. That is, no 'archive' containing multiple files is made.

Warning

This parameter is by default set to FALSE because calling wrong compression program might hang this program (e.g. if the child process is waiting for data on standard input). Enabling background data compression is normally done by setting the environment variable `AHA_ZIP_FILES=TRUE`. See commondata::system_init() for details.

Definition at line 2014 of file m_common.f90.

8.1.4.64 zip_outputs_background

`logical, parameter, public commondata::zip_outputs_background =.TRUE.`
This parameter defines if the output files are compressed in the background in the parallel mode or the program should wait for termination of the child zipping process.
Definition at line 2019 of file m_common.f90.

8.1.4.65 cmd_zip_output

`character(len=*), parameter, public commondata::cmd_zip_output = "gzip"`
This parameter defines the compression program that is executed to "zip" the data files if commondata::is_zip_outputs is enabled (TRUE). The normal compression utility is "gzip," that is found on almost any Linux/Unix system. gzip compresses each file individually and by default automatically deletes the original file. The compressed file extension is defined by commondata::zip_file_extenssion. See `http://www.gzip.org/`. Alternative compressors that are fairly widespread are `bzip2`, `lzma` and `xz`.
Definition at line 2029 of file m_common.f90.

8.1.4.66 zip_file_extenssion

`character(len=*), parameter, public commondata::zip_file_extenssion = ".gz"`
This parameter defines the compressed file extension for the external compression utility defined by the commondata::cmd_zip_output.
Definition at line 2033 of file m_common.f90.

8.1.4.67 enable_save_agents_each_timestep

`logical, parameter, public commondata::enable_save_agents_each_timestep = .FALSE.`
This parameter defines if all agents data is saved at each time step of the life cycle. See the_evolution::lifecycle_↩ preevol().

Warning

Saving large amount of data at each time step of the lifecycle is time consuming and will make calculations slower.

Definition at line 2040 of file m_common.f90.

8.1.4.68 mmdd

`character(len=:), allocatable, public, protected commondata::mmdd`
`MMDD` tag, year, month and day, used in file names and outputs. The value of the tag should be obtained only once at the start of the simulation, normally by calling the commondata::tag_mmdd() function at commondata::system_init(). It does not make much sense to generate these data tags on the fly as the simulations can be very long, several days, and so the file tags will be inconsistent.

Note

MMDD normally has a fixed length 8 as in `20161228`, see commondata::tag_mmdd(). Because it has allocatable attribute, its actual length is obtained automatically and no `trim(MMDD)` is necessary.

Definition at line 2052 of file m_common.f90.

8.1.4.69 popsize

`integer, parameter, public commondata::popsize = 10000`
Maximum population size.
Definition at line 2055 of file m_common.f90.

8.1.4.70 generations

`integer, parameter, public commondata::generations = 100`
Maximum number of generations in GA.
Definition at line 2058 of file m_common.f90.

8.1.4.71 global_generation_number_current

`integer, public commondata::global_generation_number_current`
The current global **generation number**. This is a global non fixed-parameter variable that is updated in subroutines.

Warning

There might be no guarantee that it is saved in all subroutines

Definition at line 2063 of file m_common.f90.

8.1.4.72 lifespan

`integer, parameter, public commondata::lifespan = 14000`
Number of time steps in the agent's maximum life length.
Definition at line 2066 of file m_common.f90.

8.1.4.73 preevol_tsteps

`integer, parameter, public commondata::preevol_tsteps = 560`
Number of time steps in the agent's life at the pre-evolution stage.
Definition at line 2069 of file m_common.f90.

8.1.4.74 preevol_tsteps_force_debug

`integer, parameter, public commondata::preevol_tsteps_force_debug = 280`
Number of time steps in the agent's life at the fixed fitness pre-evolution stage. This parameter **forces** a smaller fixed value that is used for debugging only. Thus, adaptive time steps calculated by the_evolution::preevol_steps_adaptive() are disabled. To enable this fixed time steps, set this parameter commondata::preevol_tsteps_force_debug_enabled to TRUE.

Warning

This is used for debugging and testing purposes only and should normally be disabled.

Definition at line 2079 of file m_common.f90.

8.1.4.75 preevol_tsteps_force_debug_enabled

`logical, parameter, public commondata::preevol_tsteps_force_debug_enabled = .FALSE.`
This parameter enables the forced smaller fixed number of time steps set by the commondata::preevol_tsteps_force_debug parameter.

Warning

This is used for debugging and testing purposes only and should normally be disabled, set to **FALSE**.

Definition at line 2085 of file m_common.f90.

8.1.4.76 lifecycle_predation_disabled_debug

`logical, parameter, public commondata::lifecycle_predation_disabled_debug = .FALSE.`
This parameter completely disables predation in the GA life cycle procedure.

Warning

Can be enabled for debugging purposes only. Normally should be set to **FALSE**.

Definition at line 2091 of file m_common.f90.

8.1.4.77 global_time_step_model_current

`integer, public commondata::global_time_step_model_current`
The current global **time step** of the model. This is a global non fixed-parameter variable that is updated in subroutines.
Definition at line 2095 of file m_common.f90.

8.1.4.78 global_frame_number

`integer, public commondata::global_frame_number`
The current global time frame. Frames are time steps within the time step defined by the commondata::global_time_step_model_curren
Definition at line 2099 of file m_common.f90.

8.1.4.79 percept_error_cv_def

`real(srp), parameter, public commondata::percept_error_cv_def = 0.01_SRP`
Default perception error in the commondata::gamma2gene() neuronal response functions. Note that this parameter defines stochastic error as the Coefficient of Variation (CV).
Definition at line 2104 of file m_common.f90.

8.1.4.80 body_length_min

`real(srp), parameter, public commondata::body_length_min = 0.2_SRP`
Minimum body length possible.
Definition at line 2118 of file m_common.f90.

8.1.4.81 body_length_max

```
real(srp), parameter, public commondata::body_length_max = 100.0_SRP
```
Maximum body length.
Definition at line 2121 of file m_common.f90.

8.1.4.82 body_mass_min

```
real(srp), parameter, public commondata::body_mass_min = 0.1_SRP
```
Minimum possible body mass, hard limit.

Note

Note that body mass is calculated from condition factor and length. = k ∗ l∗∗3

Definition at line 2126 of file m_common.f90.

8.1.4.83 init_agents_depth_is_fixed

```
logical, parameter, public commondata::init_agents_depth_is_fixed = .FALSE.
```
This parameter determines if the agents are initialised at a fixed depth at the initialisation. Agents are normally placed uniformly, the_environment::uniform(), at the initialisation. However, the depth can be fixed. In such a case they are scattered uniformly in the X and Y coordinates but with fixed depth that is set by the commondata::init_agents_depth parameter.

Warning

The agents can also be initialised with Gaussian depth. This is controlled by the parameter commondata::init_agents_depth_is_gauss. However, this parameter has precedence over Gaussian, so remember to set it FALSE if Gaussian depth initialisation is required. See the_population::member_population::place_uniform().

Definition at line 2139 of file m_common.f90.

8.1.4.84 init_agents_depth_is_gauss

```
logical, parameter, public commondata::init_agents_depth_is_gauss = .TRUE.
```
This parameter determines if the agents are initialised at a fixed depth at the initialisation. Agents are placed uniformly, the_environment::uniform(), at the initialisation. However, the depth can be a Gaussian value with the.

- mean set by the commondata::init_agents_depth
- CV set by the commondata::init_agents_depth_cv

Such a Gaussian depth patter is switched by this parameter. The other coordinates of the agents, X and Y are then uniform.

Warning

If Gaussian depth initialisation is required, the parameter commondata::init_agents_depth_is_fixed must be set to FALSE as it has a higher precedence. See the_population::member_population::place_uniform().

Definition at line 2154 of file m_common.f90.

8.1.4.85 init_agents_depth

```
real(srp), parameter, public commondata::init_agents_depth = 1833.0_SRP
```
The fixed depth at which the agents are initialised at the start of the simulation. The other coordinates are normally set the_environment::uniform() within the initialisation environment container. See the_population::member_population::place_uniform().

Note

If the parameter commondata::init_agents_depth_is_fixed is FALSE, then the agents are scattered uniformly in the whole 3D space and this parameter is ignored.

Definition at line 2163 of file m_common.f90.

8.1.4.86 init_agents_depth_cv

```
real(srp), parameter, public commondata::init_agents_depth_cv = 0.2_SRP
```

This parameter sets the Coefficient of Variation for the Gaussian depth initialisation of the agents that is controlled by commondata::init_agents_depth_is_gauss. See the_population::member_population::place_uniform().
Definition at line 2169 of file m_common.f90.

8.1.4.87 reproduction_cost_body_mass_fix

```
real(srp), parameter, public commondata::reproduction_cost_body_mass_fix = 0.2_SRP
```

The energetic cost of reproduction in terms of the agent's body mass loss.

Warning

This parameter applies to the fixed cost version of the procedure `the_body::reproduction_cost_energy_fix()`.

Definition at line 2175 of file m_common.f90.

8.1.4.88 reproduction_cost_offspring_fract_male

```
real(srp), parameter, public commondata::reproduction_cost_offspring_fract_male = 0.3_SRP
```

The component of the energetic cost of reproduction in **males** that is proportional to the total offspring mass. For details see the procedure `the_body::reproduction_cost_energy_dynamic()`.
Definition at line 2180 of file m_common.f90.

8.1.4.89 reproduction_cost_offspring_fract_female

```
real(srp), parameter, public commondata::reproduction_cost_offspring_fract_female = 1.0_SRP
```

The component of the energetic cost of reproduction in **females** that is proportional to the total offspring mass. For details see the procedure `the_body::reproduction_cost_energy_dynamic()`.
Definition at line 2186 of file m_common.f90.

8.1.4.90 reproduction_cost_body_mass_factor_male

```
real(srp), parameter, public commondata::reproduction_cost_body_mass_factor_male = 0.4_SRP
```

The component of the energetic cost of reproduction in **males** that is proportional to the agent's body mass. For details see the procedure `the_body::reproduction_cost_energy_dynamic()`.
Definition at line 2192 of file m_common.f90.

8.1.4.91 reproduction_cost_body_mass_factor_female

```
real(srp), parameter, public commondata::reproduction_cost_body_mass_factor_female = 0.1
```

The component of the energetic cost of reproduction in **females** that is proportional to the agent's body mass. For details see the procedure `the_body::reproduction_cost_energy_dynamic()`.
Definition at line 2198 of file m_common.f90.

8.1.4.92 reproduction_cost_unsuccess

```
real(srp), parameter, public commondata::reproduction_cost_unsuccess = 0.1_SRP
```

The energetic cost of unsuccessful reproduction in terms of the agent's body mass lost. This is a fraction of the **full cost of reproduction**, that is described by the REPRODUCTION_COST_BODY_MASS parameter.
Definition at line 2204 of file m_common.f90.

8.1.4.93 reproduct_body_mass_offspr_abscissa

```
real(srp), dimension(*), parameter, public commondata::reproduct_body_mass_offspr_abscissa = [
BODY_MASS_MIN, 3.0_SRP, 10.5_SRP, 12.0_SRP ]
```

The array defining the **abscissa** (X) of the nonparametric function curve that defines the relationship between the agent's body mass and the overall mass of all offspring as a fraction of the agent's body mass.

Warning

Must have the same dimensionality as commondata::reproduct_body_mass_offspr_ordinate.

Definition at line 2211 of file m_common.f90.

8.1.4.94 reproduct_body_mass_offspr_ordinate

```
real(srp), dimension(*), parameter, public commondata::reproduct_body_mass_offspr_ordinate = [
0.0_SRP, 0.1_SRP, 0.199_SRP, 0.20_SRP ]
```

The array defining the **ordinate** (Y) of the nonparametric function curve that defines the relationship between the agent's body mass and the overall mass of all offspring as a fraction of the agent's body mass. Plotting command for the interpolator:

```
htintrpl.exe [0.1 100 350 400] [0.0 0.1 0.199 0.20] [0] [nonlinear]
htintrpl.exe [0.1 3 10.5 12.0] [0.0 0.1 0.199 0.20] [0] [nonlinear]
```

Warning

Must have the same dimensionality as commondata::reproduct_body_mass_offspr_abscissa.

Definition at line 2225 of file m_common.f90.

8.1.4.95 universe_min_coord_notuse

```
real(srp), dimension(3), parameter, public commondata::universe_min_coord_notuse = [0.0_SRP,
0.0_SRP, 0.0_SRP]
```

Overall size of the global 3D universe of the model.
Physical sizes of the 3D "universe" environment for the agents' life. The *minimum* coordinates (UNIVERSE_MIN↩_COORD_NOTUSE) are all zeroes for simplicity. So here set the *maximum* coordinates vector [x,y,z] limiting the maximum environment size: UNIVERSE_WHOLE_SIZE_NOTUSE.

Warning

The dimensionality is only **3** for three-dimensional space.

Note

Universe has been deprecated in the_evolution module as it has not been used. However, the parameters are still defined as the arrays could be used in definitions of habitats, if found feasible.

Definition at line 2254 of file m_common.f90.

8.1.4.96 universe_whole_size_notuse

```
real(srp), dimension(3), parameter, public commondata::universe_whole_size_notuse = [20000.0↩
_SRP, 10000.0_SRP, 3000.0_SRP]
```
Definition at line 2256 of file m_common.f90.

8.1.4.97 dielcycles

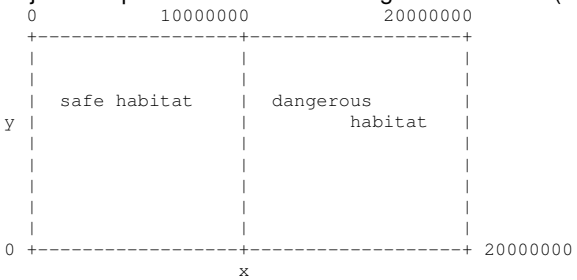
```
integer, parameter, public commondata::dielcycles = 100
```
Number of days and nights in a lifespan, DIELCYCLES=500.
Definition at line 2260 of file m_common.f90.

8.1.4.98 history_size_spatial

```
integer, parameter, public commondata::history_size_spatial = 50
```
The size of the history for spatial moving objects, i.e. how many time steps positions to remember in stack arrays.
Definition at line 2264 of file m_common.f90.

8.1.4.99 habitat_safe_min_coord

```
real(srp), dimension(3), parameter, public commondata::habitat_safe_min_coord = [0.0_SRP, 0.↩
0_SRP, 0.0_SRP]
```
Definition of the habitat spatial limits.
We define two habitats **within the global universe** (UNIVERSE_WHOLE_SIZE) of the model. They are called "The safe" and "The dangerous" habitats and primarily differ in the level of predator risk. The habitats represent two adjacent squares that form a rectangular "universe" (the universe defined by UNIVERSE_WHOLE_SIZE).



Safe habitat: 0:1000000 x 0:1000000 x 0:3000 cm (NB: units cm!)

Warning

The dimensionality is only **3** for three-dimensional space.

Definition at line 2295 of file m_common.f90.

8.1.4.100 habitat_safe_max_coord

```
real(srp), dimension(3), parameter, public commondata::habitat_safe_max_coord = [10000.0_SRP,
10000.0_SRP, 3000.0_SRP]
```
Definition at line 2297 of file m_common.f90.

8.1.4.101 habitat_danger_min_coord

```
real(srp), dimension(3), parameter, public commondata::habitat_danger_min_coord = [10000.0_↩
SRP, 0.0_SRP, 0.0_SRP]
```
Definition at line 2301 of file m_common.f90.

8.1.4.102 habitat_danger_max_coord

```
real(srp), dimension(3), parameter, public commondata::habitat_danger_max_coord = [20000.0_↩
SRP, 10000.0_SRP, 3000.0_SRP]
```
Definition at line 2303 of file m_common.f90.

8.1.4.103 predators_num_habitat_safe

```
integer, parameter, public commondata::predators_num_habitat_safe = 100
```
The **number of predators** in the **safe** habitat.
Definition at line 2309 of file m_common.f90.

8.1.4.104 predators_num_habitat_danger

```
integer, parameter, public commondata::predators_num_habitat_danger = 500
```
The **number of predators** in the **dangerous** habitat.
Definition at line 2312 of file m_common.f90.

8.1.4.105 food_abundance_habitat_safe

```
integer, parameter, public commondata::food_abundance_habitat_safe = 20000
```
The **food abundance** in the **safe** habitat.
Definition at line 2315 of file m_common.f90.

8.1.4.106 food_abundance_habitat_danger

```
integer, parameter, public commondata::food_abundance_habitat_danger = 40000
```
The **food abundance** in the **dangerous** habitat.
Definition at line 2318 of file m_common.f90.

8.1.4.107 other_risks_def

```
real(srp), parameter, public commondata::other_risks_def = 0.01_SRP
```
Default level of other mortality risks in the habitat.
Definition at line 2324 of file m_common.f90.

8.1.4.108 other_risks_habitat_safe

```
real(srp), parameter, public commondata::other_risks_habitat_safe = 0.01_SRP
```
Habitat-specific mortality risk (not linked with predation) in the **safe** habitat.
Definition at line 2328 of file m_common.f90.

8.1.4.109 other_risks_habitat_danger

```
real(srp), parameter, public commondata::other_risks_habitat_danger = 0.05_SRP
```
Habitat-specific mortality risk (not linked with predation) in the **dangerous** habitat.
Definition at line 2332 of file m_common.f90.

8.1.4.110 eggmortality_def

```
real(srp), parameter, public commondata::eggmortality_def = 0.01_SRP
```
Default level of egg mortality in the habitat.

Definition at line 2335 of file m_common.f90.

8.1.4.111 individual_mortality_risk_def

real(srp), parameter, public commondata::individual_mortality_risk_def = 0.01_SRP
Default individually-specific mortality risk. It can increase or decrease depending on various factors. The individually-specific mortality risk is normally a Gaussian variable with the variability set by the commondata::individual_mortality_risk_cv.
Definition at line 2341 of file m_common.f90.

8.1.4.112 individual_mortality_risk_cv

real(srp), parameter, public commondata::individual_mortality_risk_cv = 0.05_SRP
The coefficient of variation for Gaussian stochastic individually-specific mortality risk of the agent.
Definition at line 2345 of file m_common.f90.

8.1.4.113 predator_body_size

real(srp), parameter, public commondata::predator_body_size = 100.0_SRP
The body size of the predator. In this version all predators have the same body size set by this parameter, but can be Gaussian stochastic. Moreover, in such a case predator attack efficiency can depend on the body size, e.g. larger predators are more dangerous. compare to the agents maximum body size BODY_LENGTH_MAX=100.0
Definition at line 2357 of file m_common.f90.

8.1.4.114 predator_attack_rate_default

real(srp), parameter, public commondata::predator_attack_rate_default = 0.9_SRP
Mean rate of a single predator attack.
Definition at line 2360 of file m_common.f90.

8.1.4.115 predator_attack_rate_cv

real(srp), parameter, public commondata::predator_attack_rate_cv = 0.1_SRP
Coefficient of variation for a single predator attack among the whole population of stochastic predators.
Definition at line 2364 of file m_common.f90.

8.1.4.116 predator_attack_capture_probability_half

real(srp), parameter, public commondata::predator_attack_capture_probability_half = 0.8_SRP
The probability of capture of a fish agent by a predator at the distance equal to 1/2 of the visual range. For more details see the_environment::predator_capture_risk_calculate_fish().
Definition at line 2369 of file m_common.f90.

8.1.4.117 predator_attack_capture_probability_min

real(srp), parameter, public commondata::predator_attack_capture_probability_min = 0.1_SRP
Minimum probability of capture, e.g. at a distance exceeding the visual range. The latter assumes that the predator could detect the agent beyond the visual range and pursue it. For more details see the_environment::predator_capture_risk_calculate_fish().
Definition at line 2376 of file m_common.f90.

8.1.4.118 predator_attack_capture_prob_frz_50

```
real(srp), parameter, public commondata::predator_attack_capture_prob_frz_50 = 0.10_SRP
```

A parameter factor defining the probability of capture of an immobile (freezing) agent by a predator: interpolation ordinate for the distance equal to **0.25 of the visual range**. See the_environment::predator_capture_risk_calculate for details.
Definition at line 2383 of file m_common.f90.

8.1.4.119 predator_attack_capture_prob_frz_75

```
real(srp), parameter, public commondata::predator_attack_capture_prob_frz_75 = 0.01_SRP
```

A parameter factor defining the probability of capture of an immobile (freezing) agent by a predator: interpolation ordinate for the distance equal to **0.40 of the visual range**. See the_environment::predator_capture_risk_calculate for details.
Definition at line 2388 of file m_common.f90.

8.1.4.120 agent_can_assess_predator_attack_rate

```
logical, parameter, public commondata::agent_can_assess_predator_attack_rate = .TRUE.
```

A logical flag of whether the agents can assess the individual inherent attack rates of the predators. If yes, these inherent individual attack rates are collated into the perception object. If no, the default attack rate set by the commondata::predator_attack_rate_default parameter is used.
Definition at line 2395 of file m_common.f90.

8.1.4.121 predator_risk_group_select_index_partial

```
integer, parameter, public commondata::predator_risk_group_select_index_partial = 20
```

Sets the limit for partial indexing and ranking of **prey agents** in the visual range of the predator. The risk of predation, i.e. the probability of attack and capture of each agent in a group of agents, will be calculated individually for distance-ranked agents only up to this parameter value.
Definition at line 2402 of file m_common.f90.

8.1.4.122 predator_risk_group_dilution_ordinate

```
real(srp), dimension(*), parameter, public commondata::predator_risk_group_dilution_ordinate =
[1.0_SRP, 0.3_SRP, 0.1_SRP]
```

The array defining the ordinate grid values for the weighting nonparametric function linking the distance rank of the agent within the visual field of the predator and the weighting factor adjusting for predator confusion and predator dilution effects. The grid abscissa is calculated dynamically in the the_environment::predator_capture_risk_calculate_fish_group() procedure.

Note

Note that the middle value equal to 0.5 results in a linear function.

This command produces the function plot:

```
htintrpl.exe [1 30 60] [1 0.3 0.0] [1]
```

Definition at line 2417 of file m_common.f90.

8.1.4.123 food_item_size_default

```
real(srp), parameter, public commondata::food_item_size_default = 2.1_SRP
```

Default size of a single food item.

Note

Note that the maximum stomach capacity of the agent (the_condition::condition::maxstomcap) is 0.15 of the agent body mass (M_a). So if the average agent mass is 41.0, the maximum capacity is about 6.15. Thus, to get the food item size corresponding to such a mass (food item mass 6.15), the *size* of the food item should be:

$$s = \sqrt[3]{\frac{3 \cdot 0.15 \cdot M_a}{4\pi\rho}} \Rightarrow s = \sqrt[3]{\frac{0.45 M_a}{4\pi\rho}}.$$

Calculate: `( 0.45 * 41 / (4 * 3.1415926 * 0.1 ))^(1/3) = 2.45` In reality, it makes sense to make the mean food item size smaller than exactly 0.15 of the maxstomcap to accomodate its Gaussian variability, so more than 50% of items would fit into the stomach. An average food item mass of 0.09 of the agent mass might be better. Calculate: `( 3 * 0,09 * 41 / (4 * 3.1415926 * 0.1 ))^(1/3) = 2.065`

Definition at line 2438 of file m_common.f90.

8.1.4.124 food_item_mean_size

`real(srp), parameter, public commondata::food_item_mean_size = FOOD_ITEM_SIZE_DEFAULT`
The above is also the average size of a stochastic Gaussian food items.
Definition at line 2442 of file m_common.f90.

8.1.4.125 food_item_size_default_cv

`real(srp), parameter, public commondata::food_item_size_default_cv = 0.1_SRP`
Coefficient of variation for Gaussian food items.
Definition at line 2445 of file m_common.f90.

8.1.4.126 food_item_minimum_size

`real(srp), parameter, public commondata::food_item_minimum_size = 1.0_SRP`
The minimum size of a food item. This is the "floor" in case the stochastically generated (e.g. Gaussian) value gets zero or below.
Definition at line 2449 of file m_common.f90.

8.1.4.127 food_item_density

`real(srp), parameter, public commondata::food_item_density = 0.1_SRP`
The (physical) density of a single food item. TODO: need to parametrise!
Definition at line 2452 of file m_common.f90.

8.1.4.128 food_item_capture_prop_cost

`real(srp), parameter, public commondata::food_item_capture_prop_cost = 0.05_SRP`
The cost of the food item catching, in terms of the **food item mass** (proportional cost). So, if the agent does an unsuccessful attempt to catch a food item, the cost still applies.
Definition at line 2457 of file m_common.f90.

8.1.4.129 food_item_capture_probability

`real(srp), parameter, public commondata::food_item_capture_probability = 0.99_SRP`
The **baseline** probability that the food item is captured. See the_neurobio::food_item_capture_probability_calc().

Note

Interpolation plot command: `htintrpl.exe [0.0 0.5 1.0] [0.85, 0.68, 0.1]` (0.68=0.↩ 85∗0.8).

Definition at line 2463 of file m_common.f90.

8.1.4.130 food_item_capture_probability_min

`real(srp), parameter, public commondata::food_item_capture_probability_min = 0.1_SRP`
The **minimum** probability of capture a food item, when the item is at a distance equal to the visual range from the predator agent.
Definition at line 2467 of file m_common.f90.

8.1.4.131 food_item_capture_probability_subjective_errorr_cv

`real(srp), parameter, public commondata::food_item_capture_probability_subjective_errorr_cv = 0.1`
Subjective error assessing the food item capture probability when assessing the subjective GOS expectancies of food items. The subjective assessment value of the capture probability is equal to the objective value plus random error with the CV equal to this parameter.
Definition at line 2473 of file m_common.f90.

8.1.4.132 food_item_migrate_xy_mean

`real(srp), parameter, public commondata::food_item_migrate_xy_mean = FOOD_ITEM_SIZE_DEFAULT * 10.0_SRP`
Mean shift parameter for the local random walk movement of food items in the horizontal plane.
Definition at line 2483 of file m_common.f90.

8.1.4.133 food_item_migrate_depth_mean

`real(srp), parameter, public commondata::food_item_migrate_depth_mean = FOOD_ITEM_SIZE_DEFAULT * 100.0_SRP`
Mean shift parameter for the local random walk movement of food items in the vertical (depth) plane.
Definition at line 2488 of file m_common.f90.

8.1.4.134 food_item_migrate_xy_cv

`real(srp), parameter, public commondata::food_item_migrate_xy_cv = FOOD_ITEM_SIZE_DEFAULT_CV`
Coefficient of variation parameter for the local random walk movement of food items in the horizontal plane.
Definition at line 2493 of file m_common.f90.

8.1.4.135 food_item_migrate_depth_cv

`real(srp), parameter, public commondata::food_item_migrate_depth_cv = 0.8_SRP`
Coefficient of variation parameter for the local random walk movement of food items in the vertical (depth) plane.
Definition at line 2498 of file m_common.f90.

8.1.4.136 daylight

`real(srp), parameter, public commondata::daylight =500.0_SRP`
Maximum above-surface light intensity at midday, DAYLIGHT=500.0.

Note

Can be deterministic or stochastic.

Definition at line 2509 of file m_common.f90.

8.1.4.137 daylight_stochastic

`logical, parameter, public commondata::daylight_stochastic = .TRUE.`
Flag for stochastic daylight pattern (if TRUE) or deterministic sinusoidal (when FALSE). Check out the next parameter `DAYLIGHT_CV` for variability.
Definition at line 2513 of file m_common.f90.

8.1.4.138 daylight_cv

`real(srp), parameter, public commondata::daylight_cv =0.2_SRP`
Coefficient of variation for stochastic DAYLIGHT,.

Note

if = 0.0 then deterministic sinusoidal daylight pattern is used.

Definition at line 2517 of file m_common.f90.

8.1.4.139 beamatt

`real(srp), parameter, public commondata::beamatt =1.0_SRP`
Beam attenuation coefficient of water (m-1),BEAMATT = 1.0.

Note

See `the_environment::visual_range()` and `the_environment::srgetr()` for more details.

Definition at line 2522 of file m_common.f90.

8.1.4.140 preycontrast_default

`real(srp), parameter, public commondata::preycontrast_default = 1.0_SRP`
Inherent contrast of prey, CONTRAST =1.0.

Note

See `the_environment::visual_range()` and `the_environment::srgetr()` for more details.

Definition at line 2527 of file m_common.f90.

8.1.4.141 preyarea_default

`real(srp), parameter, public commondata::preyarea_default =3.E-6_SRP`
Area of prey (m2), PREYAREA = 3.E-6.

Note

See `the_environment::visual_range()` and `the_environment::srgetr()` for more details.

Definition at line 2532 of file m_common.f90.

8.1.4.142 viscap

```
real(srp), parameter, public commondata::viscap =1.6E6_SRP
```
Dimensionless descriptor of fish eye quality, VISCAP=1.6E6.

Note

See the_environment::visual_range() and the_environment::srgetr() for more details.

Definition at line 2537 of file m_common.f90.

8.1.4.143 eyesat

```
real(srp), parameter, public commondata::eyesat =500.0_SRP
```
Saturation parameter of eye (Ke) (uE m-2 s-1), EYESAT=500.0.

Note

See the_environment::visual_range() and the_environment::srgetr() for more details.

Definition at line 2542 of file m_common.f90.

8.1.4.144 lightdecay

```
real(srp), parameter, public commondata::lightdecay = 0.002_SRP
```
Vertical conservation of light, per depth (old code lightdecay=0.2).

Note

set = 0 if light constant with depth

Because body size of the agent is set in cm, and the environment size is also in cm, we need to scale depth appropriately, in the old code it was within the range 0:30 m or unitless?, now the range is 0:3000 cm. So the appropriate scaling factor is 0.002.

wxMaxima quick code for plotting (assuming surface light 500.0):
```
wxplot2d( 500.0*exp(-0.002 * d), [d, 0., 3000.] );
```

Definition at line 2554 of file m_common.f90.

8.1.4.145 allelerange_min

```
integer, parameter, public commondata::allelerange_min = 1
```
The minimum possible value of alleles (allele range minimum) See implementation notes on the_genome::gene::allele_val component of the the_genome::gene derived type and commondata::alleleconv() and commondata::allelesca functions.

Warning

The minimum value should **not** be zero, as it will result in *division by zero* condition in gamma2gene, (P/y)**x.

Definition at line 2576 of file m_common.f90.

8.1.4.146 allelerange_max

```
integer, parameter, public commondata::allelerange_max = 10000
```
The maximum possible value of alleles (allele range maximum) See implementation notes on the_genome::gene::allele_va component of the the_genome::gene derived type and commondata::alleleconv() and commondata::allelesca functions.
Definition at line 2582 of file m_common.f90.

8.1.4.147 allelescale_max

```
real(srp), parameter, public commondata::allelescale_max = 20.0_SRP
```

Conversion parameter that defines the scaling of the integer allele values ::ALLELERANGE_MIN to ALLELERANGE_MAX are converted to zero to this parameter value as the maximum. See allelescale() for details.
Definition at line 2587 of file m_common.f90.

8.1.4.148 additive_comps

```
integer, parameter, public commondata::additive_comps = 3
```

Number of additive allele components.
Definition at line 2590 of file m_common.f90.

8.1.4.149 mutationrate_point

```
real(srp), parameter, public commondata::mutationrate_point = 0.1_SRP
```

Mutation rate for point allele mutations.
Definition at line 2595 of file m_common.f90.

8.1.4.150 ga_mutationrate_point_max

```
real(srp), parameter, public commondata::ga_mutationrate_point_max = 0.25_SRP
```

Maximum point mutation rate in the adaptive Fixed Fitness Genetic Algorithm.
Definition at line 2599 of file m_common.f90.

8.1.4.151 mutationrate_batch

```
real(srp), parameter, public commondata::mutationrate_batch = 0.05_SRP
```

Mutation rate for point allele mutations, a whole batch of allele components.
Definition at line 2603 of file m_common.f90.

8.1.4.152 ga_mutationrate_batch_max

```
real(srp), parameter, public commondata::ga_mutationrate_batch_max = 0.1_SRP
```

Maximum batch mutation rate in the adaptive Fixed Fitness Genetic Algorithm.
Definition at line 2607 of file m_common.f90.

8.1.4.153 relocation_swap_rate

```
real(srp), parameter, public commondata::relocation_swap_rate = 0.05_SRP
```

Mutation rate for chromosome relocation, i.e. probability of a gene moving to a different position on the same chromosome: There are two kinds of relocations, swapping genes between two positions and moving a gene with subsequent shift. So we have two constants for the respective rates.
Definition at line 2614 of file m_common.f90.

8.1.4.154 relocation_shift_rate

```
real(srp), parameter, public commondata::relocation_shift_rate = 0.01_SRP
```

Definition at line 2615 of file m_common.f90.

8.1.4.155 n_chromosomes

`integer, parameter, public commondata::n_chromosomes = 6`
The number of chromosomes for the agents.

Warning

This is the base for setting dimensionality of several other parameter arrays below!

Definition at line 2622 of file m_common.f90.

8.1.4.156 len_chromosomes

`integer, dimension(n_chromosomes), parameter, public commondata::len_chromosomes = [ 6, 5, 12, 12, 12, 12 ]`
The number of alleles in each of the chromosomes. NOTE: This must be an array (vector) of the size `commondata::n_chromosomes`. We use new Fortran array constructor here to set the array values.

Warning

The dimensionality of this array must always coincide with `commondata::n_chromosomes`!

Definition at line 2629 of file m_common.f90.

8.1.4.157 max_nalleles

`integer, parameter, public commondata::max_nalleles = 12`
This parameter defines the maximum number of alleles within the chromosome It *IS NOT* intended to vary freely/independently. Used in definitions of `_GENOTYPE_PHENOTYPE` matrices, equal to the `maxval(LEN_CHROMOSOMES)`.

Note

This parameter is only used for setting the genotype x phenotype parameter matrices.

Warning

`maxval(LEN_CHROMOSOMES)` cannot be used for array declaration in many compilers, so should be set manually from values of LEN_CHROMOSOMES above. Or may be a scalar *exceeding* `maxval(LEN_CHROMOSOMES)`, in such case the extra values are padded with `.FALSE.` in `reshape` (see `reshape` in the parameter matricess below).

Definition at line 2643 of file m_common.f90.

8.1.4.158 lab_chromosomes

`character(len=*), dimension(n_chromosomes), parameter, public commondata::lab_chromosomes = [ "C_1_SEX ", "C_2_BODY", "C_3_HORM", "C_4_HUNG", "C_5_FEAR", "C_6_REPR" ]`
Set the labels of the chromosomes. NOTE, must be an array(vector)) of the size `commondata::n_chromosomes`. We use new Fortran array constructor here to set the array values.

Note

Note that the length of the string values in the array declaration below are all the same. Many compilers will issue an error otherwise. Hence, the assignment is arranged vertically.

Warning

The dimensionality of this array must always coincide with `commondata::n_chromosomes`!

Definition at line 2653 of file m_common.f90.

8.1.4.159 chromosome_ploidy

```
integer, parameter, public commondata::chromosome_ploidy = 2
```

The ploidy of the chromosome set. Can theoretically be haploid (=1), diploid (=2) or, polyploid (>2).

Warning

> The gamma2gene code does not work with *haploid* genotype commondata::chromosome_ploidy=1, in such a case there is no need to select random homologous chromosome and it is impossible to set the two parameters of the gamma2gene function.

Definition at line 2667 of file m_common.f90.

8.1.4.160 genome_recombination_ratio_mother

```
real(srp), parameter, public commondata::genome_recombination_ratio_mother = 0.8_SRP
```

The ratio of the genome that inherited from the **mother**. The other part is inherited from the father. See implementation details of the **random independent** genetic recombination procedure procedures the_genome::individual_genome::recombine().
Definition at line 2673 of file m_common.f90.

8.1.4.161 genome_crossover_fixed_mother

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::genome_crossover_fixed_mother = reshape ( [ YES,YES,YES,YES,YES,YES, YES,YES,YES,YES,YES,YES, YES,YES,YES,YES,YES,YES,
NO, NO,YES,YES,YES,YES, NO, NO,YES,YES,YES,YES, NO, NO,YES,YES,YES,YES, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

Boolean 2D matrix that determines the pattern of **fixed** chromosome **crossover**. For each chromosome, the alleles that are marked with the TRUE (YES) values are inherited from the **mother** whereas those marked FALSE (NO) are inherited from the **father**. See implementation details of the **fixed genetic crossover** procedure procedures the_genome::individual_genome::crossover().

Note

> Note that the parameter order is **reversed**: alleles are presented by rows, chromosomes by columns,just as in the genotype x phenotype boolean matrices.

Definition at line 2684 of file m_common.f90.

8.1.4.162 sex_ratio

```
real(srp), parameter, public commondata::sex_ratio = 0.5_SRP
```

Sex ratio for initialising genomes.

$$\frac{N_{males}}{N_{females}}$$

should evolve and change during the evolution
Definition at line 2711 of file m_common.f90.

8.1.4.163 sexlocus_label

```
character(len=label_length), parameter commondata::sexlocus_label ="SEX_LOCUS"
```

Labels for the sex locus alleles (gene) - vector as we don't need to label individual alleles. LABEL_LENGTH is here to avoid a GF warning.
Definition at line 2715 of file m_common.f90.

8.1.4.164 male

```
character(len=*), parameter, public commondata::male ="male"
```
Set names of the sexes – the allele labels.
Definition at line 2718 of file m_common.f90.

8.1.4.165 female

```
character(len=*), parameter, public commondata::female ="female"
```
Definition at line 2718 of file m_common.f90.

8.1.4.166 sex_genotype_phenotype

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::sex_genotype_↩
phenotype = reshape ( [ YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```
Sex definition can be implemented differently from all other traits. Here is an example of the **phenotype** x **genotype matrix** code for sex determination:

```
!> Sex definition can be implemented differently from all other traits.
!! @note   Note that the parameter order is **reversed**: alleles are
!!         presented by rows, chromosomes by columns.
logical, dimension(MAX_NALLELES,N_CHROMOSOMES), parameter,public         &
       :: SEX_GENOTYPE_PHENOTYPE = reshape (                             &
! ..........................................................................
                            [ &   !   1   2   3   4   5   6
                                  !-------------------------
                                    yes, no, no, no, no, no,   & !  1
                                     no, no, no, no, no, no,   & !  2
                                     no, no, no, no, no, no,   & !  3
                                     no, no, no, no, no, no,   & !  4
                                     no, no, no, no, no, no,   & !  5
                                     no, no, no, no, no, no,   & !  6
                                     no, no, no, no, no, no,   & !  7
                                     no, no, no, no, no, no,   & !  8
                                     no, no, no, no, no, no,   & !  9
                                     no, no, no, no, no, no,   & ! 10
                                     no, no, no, no, no, no,   & ! 11
                                     no, no, no, no, no, no ], & ! 12
                     !               -------------------------
                     ! max. n alleles: 6,  5, 12, 12, 12, 12, 12
                     [max_nalleles,n_chromosomes],[no],[2,1] )
! Note: additional reshape params:   array shape |pad | order
!                                                |with| by rows
! ..........................................................................
```

Note

Note that the parameter order is **reversed**: alleles are presented by rows, chromosomes by columns.

Definition at line 2753 of file m_common.f90.

8.1.4.167 growhorm_genotype_phenotype

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::growhorm_↩
genotype_phenotype = reshape ( [ NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```
Genotype x Phenotype matrix for **growth hormone**.
This two-dimensional array defines the phenotypic structure of the hormone objects, i.e. the correspondence between the gene objects and the trait values (produced by the sigmoid function). That is, which genes on which chromosomes contribute to the pheontypic values of the trait objects. This is a two dimensional array of the `logical` type that defines the allele and chromosome contributes to this specific trait.

Note

Note that the dimensions of the genetic structure array must fit within the `N_CHROMOSOMES x maxval(↩ LEN_CHROMOSOMES(chrom_nr))`.

Note that the parameter order is **reversed**: alleles are presented by rows, chromosomes by columns.

Definition at line 2800 of file m_common.f90.

8.1.4.168 growhorm_init

`real(srp), parameter, public commondata::growhorm_init = 0.5_SRP`
Genotype to phenotype gamma2gene initialisation value for **growth hormone**
Definition at line 2826 of file m_common.f90.

8.1.4.169 growhorm_gerror_cv

`real(srp), parameter, public commondata::growhorm_gerror_cv = 0.5_SRP`
Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).
Definition at line 2831 of file m_common.f90.

8.1.4.170 thyroid_genotype_phenotype

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::thyroid_genotype↩
_phenotype = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

Genotype x Phenotype matrix for **thyroid**.
This two-dimensional array defines the phenotypic structure of the hormone objects, i.e. the correspondence between the gene objects and the trait values (produced by the sigmoid function). That is, which genes on which chromosomes contribute to the pheontypic values of the trait objects. This is a two dimensional array of the `logical` type that defines the allele and chromosome contributes to this specific trait.

Note

Note that the dimensions of the genetic structure array must fit within the `N_CHROMOSOMES x maxval(↩ LEN_CHROMOSOMES(chrom_nr))`.

Note that the parameter order is **reversed**: alleles are presented by rows, chromosomes by columns.

Definition at line 2847 of file m_common.f90.

8.1.4.171 thyroid_init

`real(srp), parameter, public commondata::thyroid_init = 0.5_SRP`
Genotype to phenotype gamma2gene initialisation value for **thyroid**
Definition at line 2872 of file m_common.f90.

8.1.4.172 thyroid_gerror_cv

`real(srp), parameter, public commondata::thyroid_gerror_cv = 0.5_SRP`
Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).
Definition at line 2877 of file m_common.f90.

8.1.4.173 adrenaline_genotype_phenotype

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::adrenaline_↩
genotype_phenotype = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

Genotype x Phenotype matrix for **adrenaline**
Definition at line 2882 of file m_common.f90.

8.1.4.174 adrenaline_init

```
real(srp), parameter, public commondata::adrenaline_init = 0.5_SRP
```

Genotype to phenotype gamma2gene initialisation value for **adrenaline**
Definition at line 2907 of file m_common.f90.

8.1.4.175 adrenaline_gerror_cv

```
real(srp), parameter, public commondata::adrenaline_gerror_cv = 0.5_SRP
```

Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).
Definition at line 2912 of file m_common.f90.

8.1.4.176 cortisol_genotype_phenotype

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::cortisol_↩
genotype_phenotype = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

Genotype x Phenotype matrix for **cortisol**.
Definition at line 2917 of file m_common.f90.

8.1.4.177 cortisol_init

```
real(srp), parameter, public commondata::cortisol_init = 0.5_SRP
```

Genotype to phenotype gamma2gene initialisation value for **cortisol**
Definition at line 2942 of file m_common.f90.

8.1.4.178 cortisol_gerror_cv

```
real(srp), parameter, public commondata::cortisol_gerror_cv = 0.5_SRP
```

Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).
Definition at line 2947 of file m_common.f90.

8.1.4.179 testosterone_genotype_phenotype

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::testosterone_↩
genotype_phenotype = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

Genotype x Phenotype matrix for **testosterone**.

Definition at line 2952 of file m_common.f90.

8.1.4.180 testosterone_init

```
real(srp), parameter, public commondata::testosterone_init = 0.01_SRP
```
Genotype to phenotype gamma2gene initialisation value for **testosterone**
Definition at line 2977 of file m_common.f90.

8.1.4.181 testosterone_gerror_cv

```
real(srp), parameter, public commondata::testosterone_gerror_cv = 0.5_SRP
```
Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).
Definition at line 2982 of file m_common.f90.

8.1.4.182 estrogen_genotype_phenotype

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::estrogen_↩
genotype_phenotype = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```
Genotype x Phenotype matrix for ESTROGEN.
Definition at line 2987 of file m_common.f90.

8.1.4.183 estrogen_init

```
real(srp), parameter, public commondata::estrogen_init = 0.01_SRP
```
Genotype to phenotype gamma2gene initialisation value for **estrogen**
Definition at line 3012 of file m_common.f90.

8.1.4.184 estrogen_gerror_cv

```
real(srp), parameter, public commondata::estrogen_gerror_cv = 0.5_SRP
```
Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).
Definition at line 3017 of file m_common.f90.

8.1.4.185 sex_steroids_check_history

```
integer, parameter, public commondata::sex_steroids_check_history = 3
```
The number of the latest historical values that are checked for change when setting an increment of the **sex steroid** hormones.
Definition at line 3024 of file m_common.f90.

8.1.4.186 sex_steroids_increment_factor_age_curve_abscissa

```
real(srp), dimension(*), parameter, public commondata::sex_steroids_increment_factor_age_↩
curve_abscissa = [ 0.0_SRP, real(LIFESPAN, SRP)*0.25_SRP, real(LIFESPAN, SRP) ]
```
The array defining the **abscissa** (X) of the nonparametric function curve that defines the relationship between the age of the agent and the steroid increment factor for this specific age.

Note

Note that these parameters link sex steroid increment with the agent's **age**.

Warning

The LIFESPAN parameter has to be converted to the real type.

All these arrays must have the same dimensionality:

- commondata::sex_steroids_increment_factor_age_curve_abscissa
- commondata::sex_steroids_increment_factor_age_curve_ordinate
- commondata::sex_steroids_increment_factor_len_curve_abscissa
- commondata::sex_steroids_increment_factor_len_curve_ordinate

Definition at line 3040 of file m_common.f90.

8.1.4.187 sex_steroids_increment_factor_age_curve_ordinate

```
real(srp), dimension(*), parameter, public commondata::sex_steroids_increment_factor_age_↩
curve_ordinate = [ 0.0_SRP, 0.01_SRP, 0.1_SRP ]
```

The array defining the **ordinate** (Y) of the nonparametric function curve that defines the relationship between the age of the agent and the steroid increment factor for this specific age.

Warning

All these arrays must have the same dimensionality:

- commondata::sex_steroids_increment_factor_age_curve_abscissa
- commondata::sex_steroids_increment_factor_age_curve_ordinate
- commondata::sex_steroids_increment_factor_len_curve_abscissa
- commondata::sex_steroids_increment_factor_len_curve_ordinate

Definition at line 3053 of file m_common.f90.

8.1.4.188 sex_steroids_increment_factor_len_curve_abscissa

```
real(srp), dimension(*), parameter, public commondata::sex_steroids_increment_factor_len_↩
curve_abscissa = [ 0.0_SRP, BODY_LENGTH_MAX*0.2_SRP, BODY_LENGTH_MAX ]
```

The array defining the **abscissa** (X) of the nonparametric function curve that defines the relationship between the body length of the agent and the steroid increment factor for this specific length.

Note

Note that these parameters link sex steroid increment with the agent's **body length**.

Warning

All these arrays must have the same dimensionality:

- commondata::sex_steroids_increment_factor_age_curve_abscissa
- commondata::sex_steroids_increment_factor_age_curve_ordinate
- commondata::sex_steroids_increment_factor_len_curve_abscissa
- commondata::sex_steroids_increment_factor_len_curve_ordinate

Definition at line 3070 of file m_common.f90.

8.1.4.189 sex_steroids_increment_factor_len_curve_ordinate

```
real(srp), dimension(*), parameter, public commondata::sex_steroids_increment_factor_len_↩
curve_ordinate = [ 0.0_SRP, 0.01_SRP, 0.1_SRP ]
```

The array defining the **ordinate** (Y) of the nonparametric function curve that defines the relationship between the body length of the agent and the steroid increment factor for this specific length.

Warning

All these arrays must have the same dimensionality:

- commondata::sex_steroids_increment_factor_age_curve_abscissa
- commondata::sex_steroids_increment_factor_age_curve_ordinate
- commondata::sex_steroids_increment_factor_len_curve_abscissa
- commondata::sex_steroids_increment_factor_len_curve_ordinate

Definition at line 3083 of file m_common.f90.

8.1.4.190 history_size_agent_prop

```
integer, parameter, public commondata::history_size_agent_prop = 100
```

History stack size for the agent's basic properties, such as body length and body mass. Normally they are saved only for the analysis and currently not used in the perception.
Definition at line 3101 of file m_common.f90.

8.1.4.191 living_cost

```
real(srp), parameter, public commondata::living_cost = 4.0_SRP
```

Living cost in terms of food consumed. metabolic costs, p roportional to body size.
Definition at line 3105 of file m_common.f90.

8.1.4.192 mass_growth_threshold

```
real(srp), parameter, public commondata::mass_growth_threshold = 0.0001_SRP
```

A minimum body mass increment when any linear growth is possible, in units of the body mass (e.g. 0.05 = 5%)
Definition at line 3109 of file m_common.f90.

8.1.4.193 linear_growth_exponent

```
real(srp), parameter, public commondata::linear_growth_exponent = 3.0_SRP
```

Growth exponent linking linear growth and body mass growth. Based on Fulton's condition factor "cube law.".

Note

It is real and can get noninteger values due to nonisometric growth.

Definition at line 3114 of file m_common.f90.

8.1.4.194 linear_growth_hormone_increment_factor_curve_abscissa

```
real(srp), dimension(*), parameter, public commondata::linear_growth_hormone_increment_factor↩
_curve_abscissa = [ 0.0_SRP, GROWHORM_INIT, GROWHORM_INIT*3.0_SRP, GROWHORM_INIT*5.0_SRP,
GROWHORM_INIT*20.0_SRP ]
```

The array defining the **abscissa** (X) of the nonparametric function curve that defines the function linking the relationship between the growth hormone and the relative linear growth increment.

Warning

Note that these are raw values that should go via the gamma2gene fake "guess" calculation version.

Must have the same dimensionality as commondata::linear_growth_hormone_increment_factor_curve_ordinate.

Definition at line 3123 of file m_common.f90.

8.1.4.195 linear_growth_hormone_increment_factor_curve_ordinate

```
real(srp), dimension(*), parameter, public commondata::linear_growth_hormone_increment_factor↩
_curve_ordinate = [0.0_SRP, 0.6_SRP, 0.9_SRP, 0.98, 1.00_SRP]
```

The array defining the **ordinate** (Y) of the nonparametric function curve that defines the function linking the relationship between the growth hormone and the relative linear growth increment.

Warning

Must have the same dimensionality as commondata::linear_growth_hormone_increment_factor_curve_abscissa.

Definition at line 3134 of file m_common.f90.

8.1.4.196 max_stomach_capacity_def

```
real(srp), parameter, public commondata::max_stomach_capacity_def = 0.15_SRP
```

Set the maximum stomach capacity default value – fraction of the body mass available for food. Can be overriden in different agent types. Normally 15%.
Definition at line 3142 of file m_common.f90.

8.1.4.197 stomach_content_emptify_factor

```
real(srp), parameter, public commondata::stomach_content_emptify_factor = 100.0_SRP
```

Stomach content emptify factor at each step.
Stomach contents S(t) is emptied by a constant fraction each time step

$$S_t = S_{t+1}\frac{K}{\Omega}$$

, where K is the stomach content emptify factor.
Definition at line 3148 of file m_common.f90.

8.1.4.198 stomach_content_init

```
real(srp), parameter, public commondata::stomach_content_init = 0.01_SRP
```

Set average stomach capacity at birth/init in units of body weight,.

Note

it is a random Gaussian variable, this just sets the mean value, the next parameter sets coefficient of variation

Definition at line 3153 of file m_common.f90.

8.1.4.199 stomach_content_init_cv

```
real(srp), parameter, public commondata::stomach_content_init_cv = 0.05_SRP
```

Set the coefficient of variation for the stomach capacity at init.
Definition at line 3156 of file m_common.f90.

8.1.4.200 swimming_speed_cost_burst

```
real(srp), parameter, public commondata::swimming_speed_cost_burst = 0.1E-4_SRP
```

Set the weighting factor parameter of burst swimming cost in terms of the agent body size and the distance expressed in terms of the agent body lengths. In the_body::condition_cost_swimming_burst(), this parameter sets the β coefficient.
Definition at line 3162 of file m_common.f90.

8.1.4.201 cost_factor_foraging_smr

```
real(srp), parameter, public commondata::cost_factor_foraging_smr = 0.1E-5_SRP
```

Set the cost of foraging in terms of SMR.
Definition at line 3165 of file m_common.f90.

8.1.4.202 energy_genotype_phenotype

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::energy_genotype↩
_phenotype = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The initial value of the **energy reserves** at birth is genetically determined. This is the Genotype x Phenotype matrix for energy reserves.
Definition at line 3172 of file m_common.f90.

8.1.4.203 energy_init

```
real(srp), parameter, public commondata::energy_init = 0.18
```

This is the initial value of the energy reserves, non-genetic mean.
Definition at line 3197 of file m_common.f90.

8.1.4.204 energy_gerror_cv

```
real(srp), parameter, public commondata::energy_gerror_cv = 0.01
```

Genotype to phenotype initialisation, Gaussian error parameter. Coefficient of variation for the ENERGY_INIT value.
Definition at line 3201 of file m_common.f90.

8.1.4.205 body_length_genotype_phenotype

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::body_length_↩
genotype_phenotype = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The initial value of the body length at birth is genetically determined. This is the Genotype x Phenotype matrix for body length.
Definition at line 3207 of file m_common.f90.

8.1.4.206 body_length_init

```
real(srp), parameter, public commondata::body_length_init = 2.3_SRP
```

The initial value of **body length**, the average (gon-genetic).
Definition at line 3232 of file m_common.f90.

8.1.4.207 body_length_gerror_cv

```
real(srp), parameter, public commondata::body_length_gerror_cv = 0.1_SRP
```

Genotype to phenotype initialisation, Gaussian error parameter. Coefficient of variation for the BODY_LENGTH_↩ INIT value.
Definition at line 3236 of file m_common.f90.

8.1.4.208 control_unselected_genotype_phenotype

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::control_unselected↩
_genotype_phenotype = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The initial value of the **control unselected** trait. This trait is is genetically determined but is not selected or used. So it can be used to control for random genetic drift. This is the Genotype x Phenotype matrix.
Definition at line 3244 of file m_common.f90.

8.1.4.209 control_unselected_init

```
real(srp), parameter, public commondata::control_unselected_init = 0.5_SRP
```

The initial value of the **control unselected** trait that goes through the gamma2gene.
Definition at line 3270 of file m_common.f90.

8.1.4.210 control_unselected_gerror_cv

```
real(srp), parameter, public commondata::control_unselected_gerror_cv = 0.5_SRP
```

Genotype to phenotype initialisation, Gaussian error parameter. Coefficient of variation for the control unselected trait.
Definition at line 3274 of file m_common.f90.

8.1.4.211 smr_genotype_phenotype

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::smr_genotype↩
_phenotype = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The initial value of the standard metabolic rate (SMR) at birth is genetically determined. This is the Genotype x Phenotype matrix for SMR.
Definition at line 3281 of file m_common.f90.

8.1.4.212 smr_init

```
real(srp), parameter, public commondata::smr_init = 0.5_SRP
```

This is the initial value of SMR that goes through the gamma2gene.
Definition at line 3306 of file m_common.f90.

8.1.4.213 smr_gerror_cv

```
real(srp), parameter, public commondata::smr_gerror_cv = 0.5_SRP
```

Genotype to phenotype initialisation, Gaussian error parameter. Coefficient of variation for the SMR_LENGTH_↩ INIT value.
Definition at line 3310 of file m_common.f90.

8.1.4.214 smr_min

real(srp), parameter, public commondata::smr_min = 0.01_SRP
Minimum SMR value, anything lower is not allowed.
Definition at line 3313 of file m_common.f90.

8.1.4.215 swimming_cost_exponent_laminar

real(srp), parameter commondata::swimming_cost_exponent_laminar = 0.5_SRP
Default swimming cost body mass exponent parameter for **laminar** flow. See doi:10.1242/jeb.01484 (https↩ ://dx.doi.org/10.1242/jeb.01484) and the_body::condition_cost_swimming_burst() for details.
Definition at line 3318 of file m_common.f90.

8.1.4.216 swimming_cost_exponent_turbulent

real(srp), parameter commondata::swimming_cost_exponent_turbulent = 0.6_SRP
Default swimming cost body mass exponent parameter for **turbulent** flow. See doi:10.1242/jeb.01484 (https↩ ://dx.doi.org/10.1242/jeb.01484) and the_body::condition_cost_swimming_burst() for details.
Definition at line 3323 of file m_common.f90.

8.1.4.217 swimming_cost_factor_buoyancy_down

real(srp), parameter, public commondata::swimming_cost_factor_buoyancy_down = 0.01_SRP
This parameter defines the cost of the buoyancy-based locomotion as a fraction of normal laminar flow propulsion for lowering downwards.
Definition at line 3327 of file m_common.f90.

8.1.4.218 swimming_cost_factor_buoyancy_up

real(srp), parameter, public commondata::swimming_cost_factor_buoyancy_up = 0.1_SRP
This parameter defines the cost of the buoyancy-based locomotion as a fraction of normal laminar flow propulsion for lowering downwards.

Note

Note that the relative cost of buoyancy-based swimming up is higher than down because gas secretion is more slow and demanding than absorption.

Definition at line 3334 of file m_common.f90.

8.1.4.219 food_select_items_index_partial

integer, parameter, public commondata::food_select_items_index_partial = 20
Sets the limit for partial indexing and ranking of **food items** in the visual range of the agents.

Note

Calculating the distances between the agent and food items may be very slow if the number of food items is large (and every agent should do this repeatedly!!!). So we use partial indexing, we need only small number of neighbouring food items anyway.

Definition at line 3360 of file m_common.f90.

8.1.4.220 consp_select_items_index_partial

```
integer, parameter, public commondata::consp_select_items_index_partial = 20
```
Sets the limit for partial indexing and ranking of **conspecifics** in the visual range of the agent.
Definition at line 3364 of file m_common.f90.

8.1.4.221 pred_select_items_index_partial

```
integer, parameter, public commondata::pred_select_items_index_partial = 20
```
Sets the limit for partial indexing and ranking of **predators** in the visual range of the agent.
Definition at line 3368 of file m_common.f90.

8.1.4.222 individual_visual_contrast_default

```
real(srp), parameter, public commondata::individual_visual_contrast_default = 1.0_SRP
```
Inherent contrast of the **agent**, It is used in determining the visual range of an agent in perception of conspecifics, and also for assessing the agent's detectability by predator. Default Contrast of food items is set separately.
Definition at line 3374 of file m_common.f90.

8.1.4.223 history_size_perception

```
integer, parameter, public commondata::history_size_perception = HISTORY_SIZE_SPATIAL
```
Sets the size of the perception memory stack.
Definition at line 3377 of file m_common.f90.

8.1.4.224 history_size_motivation

```
integer, parameter, public commondata::history_size_motivation = HISTORY_SIZE_SPATIAL
```
Sets the size of the emotional state memory stack.
Definition at line 3386 of file m_common.f90.

8.1.4.225 light_hunger_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::light_hunger↩
_genotype_neuronal = reshape ( [ NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```
The genotype structure for **light** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.
Definition at line 3394 of file m_common.f90.

8.1.4.226 light_hunger_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::light_hunger_genotype_neuronal_gerror_cv = PERCEPT_↩
ERROR_CV_DEF
```
Gaussian perception error parameter (cv) for **light** perception effects on **hunger**.
Definition at line 3420 of file m_common.f90.

8.1.4.227 depth_hunger_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::depth_hunger↩
_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO,
```

```
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **depth** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.
Definition at line 3426 of file m_common.f90.

8.1.4.228 depth_hunger_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::depth_hunger_genotype_neuronal_gerror_cv = PERCEPT_↩
ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **depth** perception effects on **hunger**.
Definition at line 3452 of file m_common.f90.

8.1.4.229 foodcount_hunger_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::foodcount_↩
hunger_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **food items count** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.
Definition at line 3458 of file m_common.f90.

8.1.4.230 foodcount_hunger_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::foodcount_hunger_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **food items count** perception effects on **hunger**.
Definition at line 3484 of file m_common.f90.

8.1.4.231 food_mem_hunger_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::food_mem_↩
hunger_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **food items count** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.
Definition at line 3490 of file m_common.f90.

8.1.4.232 food_mem_hunger_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::food_mem_hunger_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **food items count** perception effects on **hunger**.
Definition at line 3516 of file m_common.f90.

8.1.4.233 conspcount_hunger_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::conspcount_↩
hunger_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
```

```
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **conspecifics number** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.
Definition at line 3522 of file m_common.f90.

8.1.4.234 conspcount_hunger_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::conspcount_hunger_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **conspecifics number count** perception effects on **hunger**.
Definition at line 3548 of file m_common.f90.

8.1.4.235 pred_direct_hunger_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::pred_direct_↩
hunger_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **direct predation** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.
Definition at line 3554 of file m_common.f90.

8.1.4.236 pred_direct_hunger_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::pred_direct_hunger_genotype_neuronal_gerror_cv =
PERCEPT_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **direct predation** perception effects on **hunger**.
Definition at line 3580 of file m_common.f90.

8.1.4.237 pred_meancount_hunger_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::pred_meancount↩
_hunger_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO],
[2,1] )
```

The genotype structure for **mean predator number** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.
Definition at line 3586 of file m_common.f90.

8.1.4.238 pred_meancount_hunger_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::pred_meancount_hunger_genotype_neuronal_gerror_cv =
PERCEPT_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **mean predator number** perception effects on **hunger**.
Definition at line 3612 of file m_common.f90.

8.1.4.239 predation_risk_weight_immediate

```
real(srp), parameter, public commondata::predation_risk_weight_immediate = 0.7
```

The weight of the immediately seen predators over those in the perceptual memory stack. Those in the memory then have this weight in the predation risk estimate: `1-PREDATION_RISK_WEIGHT_IMMEDIATE`. See the_neurobio::predation_risk_backend().

Note

Note that this parameter also works in appraisal functions for the fear state neuronal respones components.

Definition at line 3621 of file m_common.f90.

8.1.4.240 predation_risk_weight_memory_window

```
integer, parameter, public commondata::predation_risk_weight_memory_window = nint(HISTORY_↩
SIZE_PERCEPTION / 2.0)
```

The length of the perceptual memory window that is taken into account and weighted against the currently seen number of predators when calculating the motivation value.

Note

Note that the total memory window is `HISTORY_SIZE_PERCEPTION`.

Definition at line 3627 of file m_common.f90.

8.1.4.241 stom_hunger_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::stom_hunger_↩
genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **stomach** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.
Definition at line 3633 of file m_common.f90.

8.1.4.242 stom_hunger_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::stom_hunger_genotype_neuronal_gerror_cv = PERCEPT_↩
ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **stomach** perception effects on **hunger**.
Definition at line 3659 of file m_common.f90.

8.1.4.243 bodymass_hunger_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::bodymass_↩
hunger_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **body mass** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.
Definition at line 3665 of file m_common.f90.

8.1.4.244 bodymass_hunger_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::bodymass_hunger_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **body mass** perception effects on **hunger**.
Definition at line 3691 of file m_common.f90.

8.1.4.245 energy_hunger_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::energy_hunger↩
_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **energy reserves** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.
Definition at line 3697 of file m_common.f90.

8.1.4.246 energy_hunger_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::energy_hunger_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **energy reserves** perception effects on **hunger**.
Definition at line 3723 of file m_common.f90.

8.1.4.247 age_hunger_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::age_hunger_↩
genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **age** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.
Definition at line 3729 of file m_common.f90.

8.1.4.248 age_hunger_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::age_hunger_genotype_neuronal_gerror_cv = PERCEPT_↩
ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **age** perception effects on **hunger**.
Definition at line 3755 of file m_common.f90.

8.1.4.249 reprfac_hunger_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::reprfac_hunger↩
_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **reproductive factor** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.
Definition at line 3761 of file m_common.f90.

8.1.4.250 reprfac_hunger_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::reprfac_hunger_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **reproductive factor** perception effects on **hunger**.
Definition at line 3787 of file m_common.f90.

8.1.4.251 light_actv_avoid_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::light_actv_↩
avoid_genotype_neuronal = reshape ( [ NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **light** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.
Definition at line 3797 of file m_common.f90.

8.1.4.252 light_actv_avoid_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::light_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **light** perception effects on **fear state**.
Definition at line 3823 of file m_common.f90.

8.1.4.253 depth_actv_avoid_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::depth_actv_↩
avoid_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **depth** perception effects on **active avoidance** that goes via gamma2gene perception to neuronal response.
Definition at line 3829 of file m_common.f90.

8.1.4.254 depth_actv_avoid_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::depth_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **depth** perception effects on **fear state**.
Definition at line 3855 of file m_common.f90.

8.1.4.255 foodcount_actv_avoid_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::foodcount_actv↩
_avoid_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **food items count** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.
Definition at line 3862 of file m_common.f90.

8.1.4.256 foodcount_actv_avoid_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::foodcount_actv_avoid_genotype_neuronal_gerror_cv =
PERCEPT_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **food items count** perception effects on **fear state**.
Definition at line 3888 of file m_common.f90.

8.1.4.257 food_mem_actv_avoid_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::food_mem_actv_↩
avoid_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **food items count** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.
Definition at line 3895 of file m_common.f90.

8.1.4.258 food_mem_actv_avoid_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::food_mem_actv_avoid_genotype_neuronal_gerror_cv =
PERCEPT_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **food items count** perception effects on **fear state**.
Definition at line 3921 of file m_common.f90.

8.1.4.259 conspcount_actv_avoid_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::conspcount_↩
actv_avoid_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO],
[2,1] )
```

The genotype structure for **conspecifics number** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.
Definition at line 3928 of file m_common.f90.

8.1.4.260 conspcount_actv_avoid_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::conspcount_actv_avoid_genotype_neuronal_gerror_cv =
PERCEPT_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **conspecifics number count** perception effects on **fear state**.
Definition at line 3954 of file m_common.f90.

8.1.4.261 pred_direct_actv_avoid_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::pred_direct↩
_actv_avoid_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO],
[2,1] )
```

The genotype structure for **direct predation** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.

Definition at line 3961 of file m_common.f90.

8.1.4.262 pred_direct_actv_avoid_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::pred_direct_actv_avoid_genotype_neuronal_gerror_cv =
PERCEPT_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **direct predation** perception effects on **fear state**.
Definition at line 3987 of file m_common.f90.

8.1.4.263 pred_meancount_actv_avoid_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::pred_meancount↩
_actv_avoid_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO],
[2,1] )
```

The genotype structure for **mean predator number** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.
Definition at line 3994 of file m_common.f90.

8.1.4.264 pred_meancount_actv_avoid_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::pred_meancount_actv_avoid_genotype_neuronal_gerror_cv
= PERCEPT_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **mean predator number** perception effects on **fear state**.
Definition at line 4020 of file m_common.f90.

8.1.4.265 stom_actv_avoid_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::stom_actv_↩
avoid_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **stomach** perception effects on **active avoidance** that goes via gamma2gene perception to neuronal response.
Definition at line 4026 of file m_common.f90.

8.1.4.266 stom_actv_avoid_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::stom_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **stomach** perception effects on **fear state**.
Definition at line 4052 of file m_common.f90.

8.1.4.267 bodymass_actv_avoid_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::bodymass_actv_↩
avoid_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **body mass** perception effects on **active avoidance** that goes via gamma2gene perception to neuronal response.
Definition at line 4058 of file m_common.f90.

8.1.4.268 bodymass_actv_avoid_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::bodymass_actv_avoid_genotype_neuronal_gerror_cv =
PERCEPT_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **body mass** perception effects on **fear state**.
Definition at line 4084 of file m_common.f90.

8.1.4.269 energy_actv_avoid_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::energy_actv_↩
avoid_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **energy reserves** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.
Definition at line 4091 of file m_common.f90.

8.1.4.270 energy_actv_avoid_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::energy_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **energy reserves** perception effects on **fear state**.
Definition at line 4117 of file m_common.f90.

8.1.4.271 age_actv_avoid_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::age_actv_avoid↩
_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **age** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.
Definition at line 4124 of file m_common.f90.

8.1.4.272 age_actv_avoid_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::age_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **age** perception effects on **fear state**.
Definition at line 4150 of file m_common.f90.

8.1.4.273 reprfac_actv_avoid_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::reprfac_actv_↩
avoid_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
```

```
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **reproductive factor** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.
Definition at line 4157 of file m_common.f90.

8.1.4.274 reprfac_actv_avoid_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::reprfac_actv_avoid_genotype_neuronal_gerror_cv =
PERCEPT_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **reproductive factor** perception effects on **fear state**.
Definition at line 4183 of file m_common.f90.

8.1.4.275 light_reproduce_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::light_reproduce↩
_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **light** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.
Definition at line 4193 of file m_common.f90.

8.1.4.276 light_reproduce_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::light_reproduce_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **light** perception effects on **reproduction**.
Definition at line 4219 of file m_common.f90.

8.1.4.277 depth_reproduce_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::depth_reproduce↩
_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **depth** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.
Definition at line 4225 of file m_common.f90.

8.1.4.278 depth_reproduce_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::depth_reproduce_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **depth** perception effects on **reproduction**.
Definition at line 4251 of file m_common.f90.

8.1.4.279 foodcount_reproduce_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::foodcount_↩
reproduce_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
```

```
NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO],
[2,1] )
```

The genotype structure for **food items count** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.
Definition at line 4258 of file m_common.f90.

8.1.4.280 foodcount_reproduce_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::foodcount_reproduce_genotype_neuronal_gerror_cv =
PERCEPT_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **food items count** perception effects on **reproduction**.
Definition at line 4284 of file m_common.f90.

8.1.4.281 food_mem_reproduce_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::food_mem_↩
reproduce_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO],
[2,1] )
```

The genotype structure for **food items count** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.
Definition at line 4291 of file m_common.f90.

8.1.4.282 food_mem_reproduce_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::food_mem_reproduce_genotype_neuronal_gerror_cv =
PERCEPT_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **food items count** perception effects on **reproduction**.
Definition at line 4317 of file m_common.f90.

8.1.4.283 conspcount_reproduce_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::conspcount↩
_reproduce_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO],
[2,1] )
```

The genotype structure for **conspecifics number** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.
Definition at line 4324 of file m_common.f90.

8.1.4.284 conspcount_reproduce_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::conspcount_reproduce_genotype_neuronal_gerror_cv =
PERCEPT_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **conspecifics number count** perception effects on **reproduction**.
Definition at line 4350 of file m_common.f90.

8.1.4.285 pred_direct_reproduce_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::pred_direct_↩
reproduce_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO],
[2,1] )
```

The genotype structure for **direct predation** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.
Definition at line 4357 of file m_common.f90.

8.1.4.286 pred_direct_reproduce_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::pred_direct_reproduce_genotype_neuronal_gerror_cv =
PERCEPT_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **direct predation** perception effects on **reproduction**.
Definition at line 4383 of file m_common.f90.

8.1.4.287 pred_meancount_reproduce_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::pred_meancount↩
_reproduce_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO],
[2,1] )
```

The genotype structure for **mean predator number** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.
Definition at line 4390 of file m_common.f90.

8.1.4.288 pred_meancount_reproduce_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::pred_meancount_reproduce_genotype_neuronal_gerror_cv
= PERCEPT_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **mean predator number** perception effects on **reproduction**.
Definition at line 4416 of file m_common.f90.

8.1.4.289 stom_reproduce_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::stom_reproduce↩
_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **stomach** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.
Definition at line 4422 of file m_common.f90.

8.1.4.290 stom_reproduce_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::stom_reproduce_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **stomach** perception effects on **reproduction**.

Definition at line 4448 of file m_common.f90.

8.1.4.291 bodymass_reproduce_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::bodymass_↩
reproduce_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO],
[2,1] )
```

The genotype structure for **body mass** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.

Definition at line 4454 of file m_common.f90.

8.1.4.292 bodymass_reproduce_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::bodymass_reproduce_genotype_neuronal_gerror_cv =
PERCEPT_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **body mass** perception effects on **reproduction**.

Definition at line 4480 of file m_common.f90.

8.1.4.293 energy_reproduce_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::energy_reproduce↩
_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **energy reserves** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.

Definition at line 4487 of file m_common.f90.

8.1.4.294 energy_reproduce_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::energy_reproduce_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **energy reserves** perception effects on **reproduction**.

Definition at line 4513 of file m_common.f90.

8.1.4.295 age_reproduce_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::age_reproduce↩
_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **age** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.

Definition at line 4520 of file m_common.f90.

8.1.4.296 age_reproduce_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::age_reproduce_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **age** perception effects on **reproduction**.
Definition at line 4546 of file m_common.f90.

8.1.4.297 reprfac_reproduce_genotype_neuronal

```
logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::reprfac_reproduce↩
_genotype_neuronal = reshape ( [ NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,
NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES ], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1] )
```

The genotype structure for **age** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.
Definition at line 4553 of file m_common.f90.

8.1.4.298 reprfac_reproduce_genotype_neuronal_gerror_cv

```
real(srp), parameter, public commondata::reprfac_reproduce_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF
```

Gaussian perception error parameter (cv) for **age** perception effects on **reproduction**.
Definition at line 4579 of file m_common.f90.

8.1.4.299 modulation_appraisal_disable_all

```
logical, parameter, public commondata::modulation_appraisal_disable_all = .FALSE.
```

Disable developmental or genetic modulation of the APPRAISAL completely. If set to TRUE, **no modulation** is performed and the final motivation values are just equal to the primary modulation values in APPRAISAL.
Definition at line 4592 of file m_common.f90.

8.1.4.300 reprod_modulation_devel_agestart

```
real(srp), parameter, public commondata::reprod_modulation_devel_agestart = LIFESPAN / 2.0_SRP
```

Developmental modulation of reproductive motivation. This parameter defines the starting age (time steps) at which reproductive motivation can rise above zero (at all previous ages weight=0.0). See the_neurobio::appraisal_motivation_modulation_non_genetic().
Definition at line 4599 of file m_common.f90.

8.1.4.301 reprod_modulation_devel_agefull

```
real(srp), parameter, public commondata::reprod_modulation_devel_agefull = REPROD_MODULATION↩
_DEVEL_AGESTART + REPROD_MODULATION_DEVEL_AGESTART * 2.0_SRP / 3.0_SRP
```

Developmental modulation of reproductive motivation. This parameter defines the age (time steps) at which reproductive motivation is weighted fully (weight = 1.0). See the_neurobio::appraisal_motivation_modulation_non_genetic().
Definition at line 4606 of file m_common.f90.

8.1.4.302 reprod_modulation_devel_abscissa

```
real(srp), dimension(*), parameter commondata::reprod_modulation_devel_abscissa = [ REPROD↩
_MODULATION_DEVEL_AGESTART, REPROD_MODULATION_DEVEL_AGESTART + ( REPROD_MODULATION_DEVEL_↩
AGEFULL - REPROD_MODULATION_DEVEL_AGESTART ) / 3.0_SRP, REPROD_MODULATION_DEVEL_AGEFULL ]
```

Developmental modulation of reproductive motivation. This is the interpolation grid abscissa defining at which age reproduction motivation can have non-zero values (reproduction is possible). It defines the weighting factor applied to reproductive motivation.
It is calculated as follows. First, the first grid value *A* is set from the parameter commondata::reprod_modulation_devel_agestart (it takes the weight 0.0). Second, the last value of the abscissa grid array *B* is set from the parameter

commondata::reprod_modulation_devel_agefull (it has the weight 1.0, full reproductive factor). Third, the interval between *A* and *B* is further split into three intervals and the first point is taken as the second value of the interpolation abscissa array (assigned the weight given by commondata::reprod_modulation_devel_w2). Thus, the interval reproductive factor weighting abscissa looks like this:

```
+-------------------------------#----------+------------#----------+
1                               ^                       ^       LIFESPAN
                                A                       B

                                #-------#-------|-------#
                                        ^
                                    A+(B-A)*1/3

Resulting array abscissa: [ A,   A + (B - A) / 3,           B   ]
                ordinate: [ 0.0, REPROD_MODULATION_DEVEL_W2, 1.0 ]
```

See the_neurobio::appraisal_motivation_modulation_non_genetic().
Definition at line 4636 of file m_common.f90.

8.1.4.303 reprod_modulation_devel_w2

`real(srp), parameter, public commondata::reprod_modulation_devel_w2 = 0.1_SRP`

Developmental modulation of reproductive motivation. This parameter sets the interpolation array weight that defines how fast the reproduction motivation the_neurobio::state_reproduce is allowed to raise when the age of the agent exceeds the reproductive age. For details see the_neurobio::appraisal_motivation_modulation_non_genetic().
Definition at line 4648 of file m_common.f90.

8.1.4.304 sex_male_modulation_reproduce_genotype

`logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::sex_male_modulation_reproduce_genotype = reshape ([NO, NO, NO, NO, NO, NO, YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])`

The genotype structure for sex modulation coefficient affecting **reproduction** motivation state in **males**.
Definition at line 4653 of file m_common.f90.

8.1.4.305 sex_male_modulation_reproduce_gerror_cv

`real(srp), parameter, public commondata::sex_male_modulation_reproduce_gerror_cv = 0.1_SRP`

Gaussian error parameter (cv) for the sex modulation coefficient affecting **reproduction** motivation state in **males**.
Definition at line 4679 of file m_common.f90.

8.1.4.306 sex_female_modulation_reproduce_genotype

`logical, dimension(max_nalleles,n_chromosomes), parameter, public commondata::sex_female_modulation_reproduce_genotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])`

The genotype structure for sex modulation coefficient affecting **reproduction** motivation state in **females**.
Definition at line 4685 of file m_common.f90.

8.1.4.307 sex_female_modulation_reproduce_gerror_cv

`real(srp), parameter, public commondata::sex_female_modulation_reproduce_gerror_cv = 0.1_SRP`

Gaussian error parameter (cv) for the sex modulation coefficient affecting **reproduction** motivation state in **females**.
Definition at line 4711 of file m_common.f90.

8.1.4.308 attention_switch_hunger_light

`real(srp), parameter, public commondata::attention_switch_hunger_light = 0.0_SRP`
Baseline attention switches $\Psi_{i,j}$ control which perception components (*i*) can affect each of the motivational state (*j*). They should be defined for all combinations between the perception components (light, depth... food... predators... etc) and the motivational states. See Cognitive architecture.
These values serve as weight factors are baseline *before* attention is suppressed (the_neurobio::gos_global::attention_modulate()). They can be used for switching specific perceptual components the_neurobio::percept_components_motiv of motivation ON (1.0) or OFF (0.0) . Therefore, they should provide all possible combinations of the motivational states %state_ (as in the_neurobio::motivation) and perception components the_neurobio::percept_components_motiv.
The `ATTENTION_SWITCH_` pattern matrix for all perception components and motivation states is presented in the table below:

	HUNGER	FEAR_DEFENCE	REPRODUCE
LIGHT	0.0	1.0	0.0
DEPTH	0.0	1.0	0.0
FOOD_DIR	1.0	0.0	0.0
FOOD_MEM	1.0	0.0	0.0
CONSPEC	1.0	1.0	1.0
PRED_DIR	0.0	1.0	0.0
PREDATOR	0.0	1.0	0.0
STOMACH	1.0	0.0	0.0
BODYMASS	1.0	0.0	1.0
ENERGY	1.0	0.0	1.0
AGE	0.0	0.0	1.0
REPRFAC	0.0	0.0	1.0

Definition at line 4748 of file m_common.f90.

8.1.4.309 attention_switch_hunger_depth

`real(srp), parameter, public commondata::attention_switch_hunger_depth = 0.0_SRP`
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4750 of file m_common.f90.

8.1.4.310 attention_switch_hunger_food_dir

`real(srp), parameter, public commondata::attention_switch_hunger_food_dir = 1.0_SRP`
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4752 of file m_common.f90.

8.1.4.311 attention_switch_hunger_food_mem

`real(srp), parameter, public commondata::attention_switch_hunger_food_mem = 1.0_SRP`
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4754 of file m_common.f90.

8.1.4.312 attention_switch_hunger_conspec

`real(srp), parameter, public commondata::attention_switch_hunger_conspec = 1.0_SRP`

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4756 of file m_common.f90.

8.1.4.313 attention_switch_hunger_pred_dir

```
real(srp), parameter, public commondata::attention_switch_hunger_pred_dir = 0.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4758 of file m_common.f90.

8.1.4.314 attention_switch_hunger_predator

```
real(srp), parameter, public commondata::attention_switch_hunger_predator = 0.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4760 of file m_common.f90.

8.1.4.315 attention_switch_hunger_stomach

```
real(srp), parameter, public commondata::attention_switch_hunger_stomach = 1.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4762 of file m_common.f90.

8.1.4.316 attention_switch_hunger_bodymass

```
real(srp), parameter, public commondata::attention_switch_hunger_bodymass = 1.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4764 of file m_common.f90.

8.1.4.317 attention_switch_hunger_energy

```
real(srp), parameter, public commondata::attention_switch_hunger_energy = 1.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4766 of file m_common.f90.

8.1.4.318 attention_switch_hunger_age

```
real(srp), parameter, public commondata::attention_switch_hunger_age = 0.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4768 of file m_common.f90.

8.1.4.319 attention_switch_hunger_reprfac

```
real(srp), parameter, public commondata::attention_switch_hunger_reprfac = 0.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4770 of file m_common.f90.

8.1.4.320 attention_switch_avoid_act_light

```
real(srp), parameter, public commondata::attention_switch_avoid_act_light = 1.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4773 of file m_common.f90.

8.1.4.321 attention_switch_avoid_act_depth

```
real(srp), parameter, public commondata::attention_switch_avoid_act_depth = 1.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4775 of file m_common.f90.

8.1.4.322 attention_switch_avoid_act_food_dir

```
real(srp), parameter, public commondata::attention_switch_avoid_act_food_dir = 0.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4777 of file m_common.f90.

8.1.4.323 attention_switch_avoid_act_food_mem

```
real(srp), parameter, public commondata::attention_switch_avoid_act_food_mem = 0.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4779 of file m_common.f90.

8.1.4.324 attention_switch_avoid_act_conspec

```
real(srp), parameter, public commondata::attention_switch_avoid_act_conspec = 1.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4781 of file m_common.f90.

8.1.4.325 attention_switch_avoid_act_pred_dir

```
real(srp), parameter, public commondata::attention_switch_avoid_act_pred_dir = 1.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4783 of file m_common.f90.

8.1.4.326 attention_switch_avoid_act_predator

```
real(srp), parameter, public commondata::attention_switch_avoid_act_predator = 1.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4785 of file m_common.f90.

8.1.4.327 attention_switch_avoid_act_stomach

```
real(srp), parameter, public commondata::attention_switch_avoid_act_stomach = 0.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4787 of file m_common.f90.

8.1.4.328 attention_switch_avoid_act_bodymass

```
real(srp), parameter, public commondata::attention_switch_avoid_act_bodymass = 0.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4789 of file m_common.f90.

8.1.4.329 attention_switch_avoid_act_energy

```
real(srp), parameter, public commondata::attention_switch_avoid_act_energy = 0.0_SRP
```

Baseline attention switch, see commondata::attention_switch_hunger_light.

Definition at line 4791 of file m_common.f90.

8.1.4.330 attention_switch_avoid_act_age

```
real(srp), parameter, public commondata::attention_switch_avoid_act_age = 0.0_SRP
```
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4793 of file m_common.f90.

8.1.4.331 attention_switch_avoid_act_reprfac

```
real(srp), parameter, public commondata::attention_switch_avoid_act_reprfac = 0.0_SRP
```
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4795 of file m_common.f90.

8.1.4.332 attention_switch_reproduce_light

```
real(srp), parameter, public commondata::attention_switch_reproduce_light = 0.0_SRP
```
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4798 of file m_common.f90.

8.1.4.333 attention_switch_reproduce_depth

```
real(srp), parameter, public commondata::attention_switch_reproduce_depth = 0.0_SRP
```
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4800 of file m_common.f90.

8.1.4.334 attention_switch_reproduce_food_dir

```
real(srp), parameter, public commondata::attention_switch_reproduce_food_dir = 0.0_SRP
```
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4802 of file m_common.f90.

8.1.4.335 attention_switch_reproduce_food_mem

```
real(srp), parameter, public commondata::attention_switch_reproduce_food_mem = 0.0_SRP
```
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4804 of file m_common.f90.

8.1.4.336 attention_switch_reproduce_conspec

```
real(srp), parameter, public commondata::attention_switch_reproduce_conspec = 1.0_SRP
```
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4806 of file m_common.f90.

8.1.4.337 attention_switch_reproduce_pred_dir

```
real(srp), parameter, public commondata::attention_switch_reproduce_pred_dir = 0.0_SRP
```
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4808 of file m_common.f90.

8.1.4.338 attention_switch_reproduce_predator

```
real(srp), parameter, public commondata::attention_switch_reproduce_predator = 0.0_SRP
```
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4810 of file m_common.f90.

8.1.4.339 attention_switch_reproduce_stomach

```
real(srp), parameter, public commondata::attention_switch_reproduce_stomach = 0.0_SRP
```
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4812 of file m_common.f90.

8.1.4.340 attention_switch_reproduce_bodymass

```
real(srp), parameter, public commondata::attention_switch_reproduce_bodymass = 1.0_SRP
```
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4814 of file m_common.f90.

8.1.4.341 attention_switch_reproduce_energy

```
real(srp), parameter, public commondata::attention_switch_reproduce_energy = 1.0_SRP
```
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4816 of file m_common.f90.

8.1.4.342 attention_switch_reproduce_age

```
real(srp), parameter, public commondata::attention_switch_reproduce_age = 1.0_SRP
```
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4818 of file m_common.f90.

8.1.4.343 attention_switch_reproduce_reprfac

```
real(srp), parameter, public commondata::attention_switch_reproduce_reprfac = 1.0_SRP
```
Baseline attention switch, see commondata::attention_switch_hunger_light.
Definition at line 4820 of file m_common.f90.

8.1.4.344 attention_modulation_curve_abscissa

```
real(srp), dimension(*), parameter, public commondata::attention_modulation_curve_abscissa
=[0.0_SRP, 0.3_SRP, 0.5_SRP, 1.0_SRP]
```
The array defining the **abscissa** (X) of the nonparametric function that defines the **attention modulation curve** by the current Global Organismic State (GOS).

Warning

Must have the same dimensionality as commondata::attention_modulation_curve_ordinate.

Definition at line 4831 of file m_common.f90.

8.1.4.345 attention_modulation_curve_ordinate

```
real(srp), dimension(*), parameter, public commondata::attention_modulation_curve_ordinate
=[1.0_SRP, 0.98_SRP, 0.9_SRP, 0.0_SRP]
```
The array defining the **ordinate** (Y) of the nonparametric function that defines the **attention modulation curve** by the current Global Organismic State (GOS).

Warning

Must have the same dimensionality as commondata::attention_modulation_curve_abscissa.

Definition at line 4839 of file m_common.f90.

8.1.4.346 motivation_compet_threshold_curve_abscissa

```
real(srp), dimension(*), parameter, public commondata::motivation_compet_threshold_curve_↩
abscissa = [0.0_SRP, 0.2_SRP, 0.60_SRP, 0.80_SRP, 0.90_SRP, 1.0_SRP, 1.1_SRP]
```

The array defining the **abscissa** (X) of the nonparametric function curve that defines the threshold for motivation competition in GOS.

Warning

Must have the same dimensionality as commondata::motivation_compet_threshold_curve_ordinate.

Definition at line 4849 of file m_common.f90.

8.1.4.347 motivation_compet_threshold_curve_ordinate

```
real(srp), dimension(*), parameter, public commondata::motivation_compet_threshold_curve_↩
ordinate = [1.0_SRP, 0.3_SRP, 0.04_SRP, 0.01_SRP, 0.001_SRP, 0.0_SRP, 0.0_SRP]
```

The array defining the **ordinate** (Y) of the nonparametric function curve that defines the threshold for motivation competition in GOS.

Warning

Must have the same dimensionality as commondata::motivation_compet_threshold_curve_abscissa.

Definition at line 4857 of file m_common.f90.

8.1.4.348 arousal_gos_dissipation_factor

```
real(srp), parameter, public commondata::arousal_gos_dissipation_factor = 0.5_SRP
```

Spontaneous arousal dissipation level when a simple **fixed** dissipation factor pattern is used. At each step, `gos↩_arousal` is reduced by a constant factor, AROUSAL_GOS_DISSIPATION_FACTOR` (e.g. reduced by 0.5) independently on the current GOS time step.
Definition at line 4865 of file m_common.f90.

8.1.4.349 arousal_gos_dissipation_nonpar_abscissa

```
real(srp), dimension(*), parameter, public commondata::arousal_gos_dissipation_nonpar_abscissa
= [ 1.0, 2.00, 5.00, 10.0, 15.0, 18.0, 20.0 ]
```

This is the array defining the **abscissa** (X) of the nonparametric spontaneous arousal dissipation factor function involving polynomial (or linear) interpolation is used.

Warning

Must have the same dimensionality as commondata::arousal_gos_dissipation_nonpar_ordinate.

Definition at line 4872 of file m_common.f90.

8.1.4.350 arousal_gos_dissipation_nonpar_ordinate

```
real(srp), dimension(*), parameter, public commondata::arousal_gos_dissipation_nonpar_ordinate
= [ 1.0, 0.98, 0.80, 0.40, 0.22, 0.18, 0.17 ]
```

This is the array defining the **ordinate** (Y) of the nonparametric spontaneous arousal dissipation factor function involving polynomial (or linear) interpolation is used.

Warning

Must have the same dimensionality as commondata::arousal_gos_dissipation_nonpar_abscissa.

Definition at line 4881 of file m_common.f90.

8.1.4.351 global_rescale_maximum_motivation

`real(srp), public commondata::global_rescale_maximum_motivation`
Global maximum sensory information that is updated for the whole population of agents.

Note

This roughly corresponds to the `MaxPercept` array in the G1 model.

Definition at line 4888 of file m_common.f90.

8.1.4.352 history_size_behaviours

`integer, parameter, public commondata::history_size_behaviours = HISTORY_SIZE_SPATIAL`
The size of the behaviour labels history stack, i.e. for how many time steps should the stack remember record the behaviour labels.
Definition at line 4897 of file m_common.f90.

8.1.4.353 probability_reproduction_base_factor

`real(srp), parameter, public commondata::probability_reproduction_base_factor = 0.90`
Default weighting factor for the baseline probability of successful reproduction φ. See implementation details for the function the_neurobio::reproduce_do_probability_reproduction_calc().
Definition at line 4902 of file m_common.f90.

8.1.4.354 probability_reproduction_delta_mass_abscissa

`real(srp), dimension(*), parameter, public commondata::probability_reproduction_delta_mass_↩ abscissa = [0.5_SRP, 1.0_SRP, 2.0_SRP]`
Interpolation grid **abscissa** for the body mass ratio factor that scales the probability of reproduction. For details see the_neurobio::reproduce_do_probability_reproduction_calc() procedure. Commands (template) to produce interpolation plots:

```
htintrpl.exe [0.5 1  2] [0 1 1.8] 0.2
```

Warning

Must have the same dimensionality as commondata::probability_reproduction_delta_mass_ordinate.

Definition at line 4913 of file m_common.f90.

8.1.4.355 probability_reproduction_delta_mass_ordinate

`real(srp), dimension(*), parameter, public commondata::probability_reproduction_delta_mass_↩ ordinate = [0.0_SRP, 1.0_SRP, 1.8_SRP]`
Interpolation grid **ordinate** for the body mass ratio factor that scales the probability of reproduction. For details see the_neurobio::reproduce_do_probability_reproduction_calc() procedure. Commands (template) to produce interpolation plots:

```
htintrpl.exe [0.5 1  2] [0 1 1.8] 0.2
```

Warning

Must have the same dimensionality as commondata::probability_reproduction_delta_mass_abscissa.

Definition at line 4926 of file m_common.f90.

8.1.4.356 sex_steroids_reproduction_threshold

`real(srp), parameter, public commondata::sex_steroids_reproduction_threshold = 1.3_SRP`
This parameter defines the threshold of the current gonadal steroids level that should exceed the baseline value determined by the genome, for reproduction to be possible.

Note

Note that this parameter should normally exceed 1.0

Definition at line 4934 of file m_common.f90.

8.1.4.357 walk_random_distance_default_factor

`real(srp), parameter, public commondata::walk_random_distance_default_factor = 10.0_SRP`
The weighting factor used in calculation of the default random walk distance, in terms of the agent's body length.
Definition at line 4938 of file m_common.f90.

8.1.4.358 walk_random_distance_stochastic_cv

`real(srp), parameter, public commondata::walk_random_distance_stochastic_cv = 0.5_SRP`
The coefficient of variation of the distance for stochastic Gaussian random walk (distance is in terms of the agent's body length). Note that for deterministic walk, cv is zero.
Definition at line 4944 of file m_common.f90.

8.1.4.359 walk_random_food_gain_hope

`real(srp), parameter, public commondata::walk_random_food_gain_hope = 4.0_SRP`
The maximum walk distance, **in units of the average distance to food items** in the current perception object, when the expected food gain is calculated on the bases of the current food availability, not using the the_behaviour::hope() function mechanism. If the average walk distance exceeds this value, the expectancy is based on the the_behaviour::hope() function.
Definition at line 4952 of file m_common.f90.

8.1.4.360 walk_random_food_gain_hope_agentl

`real(srp), parameter, public commondata::walk_random_food_gain_hope_agentl = 100.0_SRP`
The maximum walk distance, **in units of the agent body length**, when the expected food gain is calculated on the bases of the current food availability, not using the the_behaviour::hope() function mechanism. If the average walk distance exceeds this value, the expectancy is based on the the_behaviour::hope() function.

Note

This parameter is used for switching between the food gain calculation methods based on current perception or hope function if the agent has no food items in perception object. Normally, if there are food items, the commondata::walk_random_food_gain_hope is used.

Definition at line 4964 of file m_common.f90.

8.1.4.361 walk_random_pred_risk_hope_agentl

```
real(srp), parameter, public commondata::walk_random_pred_risk_hope_agentl = 150.0_SRP
```

The maximum walk distance, **in units of the agent body length**, when the expected predation risk is calculated on the basis of the current perception value, not using the the_behaviour::hope() function mechanism. If the average walk distance exceeds this value, the risk expectancy is based on the the_behaviour::hope() function.
Definition at line 4971 of file m_common.f90.

8.1.4.362 walk_random_vertical_shift_ratio

```
real(srp), parameter, public commondata::walk_random_vertical_shift_ratio = 0.5_SRP
```

The ratio of the vertical to main horizontal shift parameters of the agent's Gaussian random walk. Random walk is done in the "2.5D" mode (the_environment::spatial_moving::rwalk25d()), i.e. with separate parameters for the main horizontal shift and the vertical depth shift. This is done to avoid a potentially too large vertical displacement of the agent during the movement. Thus, the vertical shift distance should normally be smaller than the horizontal shift. The difference between the main horizontal and (the smaller) vertical shifts is defined by this parameter. For example, if it is equal to 0.5, then the vertical depth shift is 0.5 of the main horizontal shift. See the_behaviour::walk_random_do_execute() for more details.
Definition at line 4984 of file m_common.f90.

8.1.4.363 walk_random_vertical_shift_cv_ratio

```
real(srp), parameter, public commondata::walk_random_vertical_shift_cv_ratio = 1.0_SRP
```

The ratio of the vertical to the main horizontal coefficients of variation for the **vertical** depth distance in the stochastic Gaussian random walk of the agent. Should normally be equal to the main default value set by commondata::walk_random_distance_stochastic_cv. That is 1.0.
Definition at line 4990 of file m_common.f90.

8.1.4.364 walk_random_food_hope_abscissa

```
real(srp), dimension(*), parameter, public commondata::walk_random_food_hope_abscissa = [ 0.0↩
_SRP, 1.0_SRP, 3.5_SRP ]
```

This parameter defines the hope function for calculating the food perception expectancy in the the_behaviour::walk_random behaviour. This is the abscissa for the hope function grid array. Plotting: `htintrpl.exe [0 1 3.5 ] [2, 1, 0]`. See the_behaviour::walk_random_do_this().

Note

Note that the "maximum hope" value for normal random walk is smaller than in the hope function grid for the migration behaviour commondata::migrate_food_gain_maximum_hope. Thus, the maximum incentive for movement is lower here.

Warning

Must have the same dimensionality as commondata::walk_random_food_hope_ordinate.

Definition at line 5003 of file m_common.f90.

8.1.4.365 walk_random_food_hope_ordinate

```
real(srp), dimension(*), parameter, public commondata::walk_random_food_hope_ordinate = [ 2.0↩
_SRP, 1.0_SRP, 0.0_SRP ]
```

This parameter defines the hope function for calculating the food perception expectancy in the the_behaviour::walk_random behaviour. This is the ordinate for the hope function grid array. Plotting: `htintrpl.exe [0 1 3.5 ] [2, 1, 0]`. See the_behaviour::walk_random_do_this().

Note

Note that the "zero hope" value for normal random walk is higher (non-zero) than in the hope function grid for the migration behaviour. Thus, the maximum incentive for movement is lower here.

Warning

Must have the same dimensionality as commondata::walk_random_food_hope_abscissa.

Definition at line 5017 of file m_common.f90.

8.1.4.366 approach_offset_default

```
real(srp), parameter, public commondata::approach_offset_default = TOLERANCE_HIGH_DEF_SRP
```

Default offset for approach, offset is the difference between the approaching agent and the target object.
Definition at line 5022 of file m_common.f90.

8.1.4.367 approach_conspecfic_dilute_general_risk

```
real(srp), parameter, public commondata::approach_conspecfic_dilute_general_risk = 0.5_SRP
```

Multiplication factor for the general risk of predation used when the agent evaluates the approach to a target conspecific.
Definition at line 5027 of file m_common.f90.

8.1.4.368 approach_conspecfic_dilute_adjust_pair_behind

```
real(srp), parameter, public commondata::approach_conspecfic_dilute_adjust_pair_behind = 0.5↩
_SRP
```

Multiplication factor for subjective assessment of the direct risk of predation when the actor agent moves behind the target conspecific, i.e. when the distance between the agent and predator is going to become longer than the distance between the target conspecific and the agent. See the_behaviour::approach_conspecifics_do_this() for details.
Definition at line 5035 of file m_common.f90.

8.1.4.369 approach_food_gain_compet_factor_abscissa

```
real(srp), dimension(*), parameter, public commondata::approach_food_gain_compet_factor_↩
abscissa = [ 0.00_SRP, 0.10_SRP, 1.00_SRP, 1.50_SRP ]
```

The grid **abscissa** defining the nonparametric relationship that determines the expected food gain for the "approach conspecifics" behaviour (the_behaviour::approach_conspec class). The function is a weighting factor depending on the ratio of the agent body mass to the target conspecific body mass, for the baseline expected food gain.

```
htintrpl.exe [ 0 0.1 1 1.5 ] [ 0 0.01 0.5 1 ]
```

Note

The (last) maximum value of the grid defines the body mass ratio that guarantees 100% expectancy of winning of competition for food against the target conspecific. For example, the value of 1.5 means that an agent is guaranteed to get the whole baseline expected food gain if its body weight is 1.5 of the target conspecific. See the_behaviour::approach_conspecifics_do_this() for details.

Warning

Must have the same dimensionality as commondata::approach_food_gain_compet_factor_ordinate.

Definition at line 5055 of file m_common.f90.

8.1.4.370 approach_food_gain_compet_factor_ordinate

```
real(srp), dimension(*), parameter, public commondata::approach_food_gain_compet_factor_↩
ordinate = [ 0.00_SRP, 0.01_SRP, 0.50_SRP, 1.00_SRP ]
```

The grid **ordinate** defining the nonparametric relationship that determines the expected food gain for the "approach conspecifics" behaviour (the_behaviour::approach_conspec class). The function is a weighting factor depending on the ratio of the agent body mass to the target conspecific body mass, for the baseline expected food gain.

```
htintrpl.exe [ 0 0.1 1 1.5 ] [ 0 0.01 0.5 1 ]
```

See the_behaviour::approach_conspecifics_do_this() for details.

Warning

Must have the same dimensionality as commondata::approach_food_gain_compet_factor_abscissa.

Definition at line 5070 of file m_common.f90.

8.1.4.371 dist_expect_food_uncertain_fact

```
real(srp), parameter, public commondata::dist_expect_food_uncertain_fact = 0.7_SRP
```

The weighting factor for the distance to the expected food item if the actual distance is uncertain (e.g. no food items currently in perception). See the_behaviour::walk_random_motivations_expect().
Definition at line 5077 of file m_common.f90.

8.1.4.372 history_perception_window_pred

```
real(srp), parameter, public commondata::history_perception_window_pred = 0.3_SRP
```

The size of the memory window that is used in the assessment of predation risk, as a portion of the commondata::history_size_perception. See the_behaviour::walk_random_do_this() and the_behaviour::walk_random_motivations_ex
Definition at line 5083 of file m_common.f90.

8.1.4.373 history_perception_window_food

```
real(srp), parameter, public commondata::history_perception_window_food = 0.3_SRP
```

The size of the memory window that is used in the assessment of food gain, as a portion of the commondata::history_size_perception. See the_behaviour::walk_random_do_this() and the_behaviour::walk_random_motivations_ex
Definition at line 5089 of file m_common.f90.

8.1.4.374 escape_dart_distance_default_factor

```
real(srp), parameter, public commondata::escape_dart_distance_default_factor = 1.5_SRP
```

The weighting factor used in calculation of the default escape distance. The escape distance is equal to the visibility range of the predator multiplied by this factor. Therefore, it should normally exceed 1.0. Otherwise, the escaping object is still within the visibility range of the predator after the escape. See the_behaviour::escape_dart_do_this() for more details.
Definition at line 5097 of file m_common.f90.

8.1.4.375 escape_dart_distance_default_stoch_cv

```
real(srp), parameter, public commondata::escape_dart_distance_default_stoch_cv = 0.5_SRP
```

For stochastic escape, this parameter determines the coefficient of variation of the escape walk. See the_behaviour::escape_dart_do_this() for more details.
Definition at line 5102 of file m_common.f90.

8.1.4.376 up_down_walk_step_stdlength_factor

`real(srp), parameter, public commondata::up_down_walk_step_stdlength_factor = 4.0_SRP`
The default size of the up and down walks performed by the GO_DOWN_DEPTH and GO_UP_DEPTH, see the_behaviour::go_down_depth and the_behaviour::go_up_depth classes as well as the_behaviour::go_down_do_this() and the_behaviour::go_up_do_this() methods.
Definition at line 5109 of file m_common.f90.

8.1.4.377 migrate_dist_max_step

`real(srp), parameter, public commondata::migrate_dist_max_step = 800.0_SRP`
The maximum distance (in units of the agent body length) a migrating agent can pass for a single time step of the model. This is basically limited by (an implicit) maximum speed of the agent, in terms of its body length. This parameter sets the limit on the length of a single migration bout.
Definition at line 5116 of file m_common.f90.

8.1.4.378 migrate_random_max_dist_target

`real(srp), parameter, public commondata::migrate_random_max_dist_target = 10.0_SRP`
Default maximum distance towards the target environment (in units of the agent's body size) when the agent could emigrate into this target environment. See the_behaviour::behaviour_do_migrate_random() for details.
Definition at line 5121 of file m_common.f90.

8.1.4.379 migrate_dist_penetrate_offset

`real(srp), parameter, public commondata::migrate_dist_penetrate_offset = 1.0_SRP`
The offset, in terms of the body length of the actor agent, for initial penetrating into the target environment when the agent is migrating into this environment. See the_environment::migrate_do_this().
Definition at line 5126 of file m_common.f90.

8.1.4.380 migrate_food_gain_maximum_hope

`real(srp), parameter, public commondata::migrate_food_gain_maximum_hope = 2.0_SRP`
This parameter defines the hope function for calculating the food gain expectancy in the migration behaviour. This is the maximum value of the hope function that is achieved at zero ratio of the old to new food gain memory values. Plotting: `htintrpl.exe [0 1 3.5] [2 1 0]`. See the_behaviour::migrate_do_this().
Definition at line 5133 of file m_common.f90.

8.1.4.381 migrate_food_gain_ratio_zero_hope

`real(srp), parameter, public commondata::migrate_food_gain_ratio_zero_hope = 3.5_SRP`
This parameter defines the hope function for calculating the food gain expectancy in the migration behaviour. This is the maximum ratio of the old to new food gain memory values that leads to virtually zero value of the hope function. Plotting: `htintrpl.exe [0 1 3.5] [2 1 0]`. See the_behaviour::migrate_do_this().
Definition at line 5140 of file m_common.f90.

8.1.4.382 migrate_predator_maximum_hope

`real(srp), parameter, public commondata::migrate_predator_maximum_hope = 2.0_SRP`
This parameter defines the hope function for calculating the general predation risk expectancy in the migration behaviour. This is the maximum value of the hope function that is achieved at zero ratio of the old to new predation values in the memory stack. Plotting: `htintrpl.exe [0 1 3.5] [2 1 0]`. See the_behaviour::migrate_do_this().
Definition at line 5148 of file m_common.f90.

8.1.4.383 migrate_predator_zero_hope

```
real(srp), parameter, public commondata::migrate_predator_zero_hope = 3.5_SRP
```

This parameter defines the hope function for calculating the general predation risk expectancy in the migration behaviour. This is the maximum ratio of the old to new predation values in the memory stack that leads to virtually zero value of the hope function. Plotting: `htintrpl.exe [0 1 3.5] [2 1 0]`. See the_behaviour::migrate_do_this().
Definition at line 5156 of file m_common.f90.

8.1.4.384 behav_walk_step_stdlen_static

```
real(srp), dimension(*), parameter, public commondata::behav_walk_step_stdlen_static = [ 1.0_↩
SRP, 10.0_SRP, 25.0_SRP, 50.0_SRP, 100.0_SRP ]
```

This parameter array defines the repertoire of predetermined static walk step sizes, in units of the agent's body length, for the the_behaviour::walk_random behavioural unit as executed in the the_behaviour::behaviour::walk_random class level. See the_behaviour::behaviour::select() method for details.
Definition at line 5163 of file m_common.f90.

8.1.4.385 behav_go_up_down_step_stdlen_static

```
real(srp), dimension(*), parameter, public commondata::behav_go_up_down_step_stdlen_static = [
10.0_SRP, 20.0_SRP, 50.0_SRP, 75.0_SRP, 100.0_SRP ]
```

This parameter array defines the step sizes, in units of the agent's body length, for the the_behaviour::go_down_depth and the_behaviour::go_up_depth behavioural unit as executed in the the_behaviour::behaviour::depth_down and the_behaviour::behaviour::depth_up class level(s). See the_behaviour::behaviour::select() method for details.
Definition at line 5171 of file m_common.f90.

8.1.4.386 ga_reproduce_pr

```
real(srp), parameter, public commondata::ga_reproduce_pr = 0.05_SRP
```

Percentage of the best reproducing agents in the pre-evolution phase.
Definition at line 5185 of file m_common.f90.

8.1.4.387 ga_reproduce_n

```
integer, parameter, public commondata::ga_reproduce_n = int(POPSIZE * GA_REPRODUCE_PR)
```

Upper limit on the number of reproducing individuals in the fixed-fitness pre-evolution phase.
Definition at line 5189 of file m_common.f90.

8.1.4.388 ga_fitness_dead

```
integer, parameter, public commondata::ga_fitness_dead = 400000000
```

Fitness value ascribed to dead agent in pre-evol. See the_individual::individual_agent::fitness_calc(). Also note that huge(integer)) = 2147483647.
Definition at line 5194 of file m_common.f90.

8.1.4.389 ga_fitness_select

```
integer, parameter, public commondata::ga_fitness_select = 900
```

Fitness threshold for the inclusion of the agent into the reproducing elite group.
Definition at line 5198 of file m_common.f90.

8.1.4.390 ga_reproduce_min_prop

```
real(srp), parameter, public commondata::ga_reproduce_min_prop = 0.05_SRP
```

Minimum proportion of reproducing agents, but note that the number of number reproducers cannot be smaller than the absolute minimum commondata::ga_reproduce_n_min. See the_population::population::ga_reproduce_max(). Definition at line 5204 of file m_common.f90.

8.1.4.391 ga_reproduce_n_min

```
integer, parameter, public commondata::ga_reproduce_n_min = 20
```

Absolute minimum number of reproducing agents in the adaptive GA procedure. See the_population::population::ga_reproduce_max() Definition at line 5208 of file m_common.f90.

8.2 file_io Module Reference

Definition of high level file objects.

Data Types

- type file_handle

  FILE_HANDLE is the basic file handle object. It provides an unitary object oriented interface for operations with any supported file types.

Functions/Subroutines

- logical function file_operation_last_is_success (this)

  Get the success or error status of the latest file operation. ***Example:***
- character(len=:) function, allocatable file_hangle_get_name_string (this)

  Get the file name associated with the file handle. If the file name is (yet) undefined, the latest operation success flag (see file_io::is_success()) is FALSE. ***Example:***
- integer function file_object_get_associated_unit (this)

  A Low level function to get the Fortran unit number associated with the file handle object.
- logical function file_object_format_is_csv (this)

  Check if the file format is CSV.
- logical function file_object_format_is_txt (this)

  Check if the file format is CSV.
- subroutine csv_open_write_this (this, name, format)

  This is an object oriented wrapper for CSV_OPEN_WRITE(). For details see CSV_OPEN_WRITE.
- subroutine csv_close_this (this)

  This is an object oriented wrapper for CSV_CLOSE(). For details see CSV_CLOSE.
- subroutine csv_header_line_write_this (this, header)

  This is an object oriented wrapper for CSV_HEADER_WRITE(). See CSV_HEADER_WRITE for details.
- subroutine csv_record_string_write_this (this, csv_record)

  Physically write a single string CSV data record to the file. See CSV_RECORD_WRITE ***Example:***

Public enumeration constants defining supported file types

Define the file types that are supported by this module.

- enum
- @, public undefined
- @, public format_csv
- @, public format_txt

8.2.1 Detailed Description

Definition of high level file objects.

8.2.2 FILE_IO module

This module defines input and output to external files in various formats. The main format for output files for numerical data is CSV. It is useful for one-dimensional vectors and two-dimensional matrices. More complex data structures can be output in other formats (to be implemented) such as binary, XML or HDF. This module provides an unitary object-oriented interface to all file types and objects.
Current status: Only CSV and plain text (TXT) files are implemented so far. And even here, only the file object is implemented in the object oriented style, record (string) is used exactly as in the CSV_IO in HEDTOOLS.

8.2.2.1 CSV format

Notably, standard HEDTOOLS whole array procedure CSV_MATRIX_WRITE can save a single vector or 2D matrix into a separate CSV file.
Example:

```
! File handle object identifies a specific file.
type(FILE_HANDLE) :: test_file
! String variable that is used to build each record (row)
! it must fit the whole record into its length;
character(len=:), allocatable :: record_string
! Open the file for writing
call test_file%open_write( "test_file.csv", format_csv )
! Write optional header.
call test_file%header_write("Header is the first line of the file")
! The current record must be cleared before it is built. Note that
! if the record string is allocatable, it must be whole blank.
record_string=repeat(" ", max_len)
! Append values to the first record. The first record is
! here the variable names.
call csv_record_append( record_string, ["No", "V1", "V2", "V3"] )
! Write this record 'record_string' physically to the disk.
call test_file%record_write(record_string)
! Now write the data beyond the first row.
do i=1, 100
  record_string=repeat(" ", max_len)  ! clean each new record string
  ! Append values of various types to the current
  ! record string 'record_string'
  call csv_record_append( record_string, i )
  call csv_record_append( record_string, [ 1.1, 2.2 ] )
  call csv_record_append( record_string, 3.3 )
  ! Once the record is built, write it to the disk.
  call test_file%record_write(record_string)
end do
! Close file at the end.
call test_file%close()
```

See code of the the_population::population_save_data_all_agents_csv() for another example of writing data to CSV file. But note that these codes use the standard non-object-oriented procedures from HEDTOOLS, not these wrappers.

8.2.2.2 Plain text (TXT) format

The file handler object file_io::file_handle defined in this module can be easily used to write arbitrary plain text files. Here is an example.
Example:

```
! File handle object identifies a specific file.
type(FILE_HANDLE) :: test_file
! Open the file for writing
call test_file%open_write( "test_file.txt", format_txt )
! Write an arbitrary row of data to the file, character text string:
call test_file%record_write("This is a test string to output")
! Standard Fortran intrinsic 'write' can be combined with the 'get_unit()'
! accessor function for unit.
write( test_file%get_unit(), * ) "Raw string 1, success=", test_file%is_success()
write( test_file%get_unit(), * ) "Raw string 2, success=", test_file%is_success()
write( test_file%get_unit(), * ) "Raw string 3, success=", test_file%is_success()
! Close file at the end.
call test_file%close()
```

Thus, the wrappers implemented in this unit allow to use unitary file handler object to work with specific files, even though they are not fully object oriented. Using a single file handle is simpler and more understandable than different Fortran file identifiers (file name, unit).

The logger commondata::logger_init() is the standard normal method to report everything in the model during the runtime. Therefore, using separate plain text file(s) to output any reports should be very rare if needed at all.

8.2.2.2.1 Accessibility of objects

By default, all objects in `FILE_IO` are *private*.

8.2.3 Enumeration Type Documentation

8.2.3.1 anonymous enum

```
anonymous enum  [private]
```

Definition at line 120 of file m_fileio.f90.

8.2.4 Function/Subroutine Documentation

8.2.4.1 file_operation_last_is_success()

```
logical function file_io::file_operation_last_is_success (
            class(file_handle), intent(inout) this )
```

Get the success or error status of the latest file operation. **Example:**

```
if ( data_file%is_success() ) then
```

Returns

TRUE if the latest file operation was successful, FALSE otherwise.

Definition at line 244 of file m_fileio.f90.

8.2.4.2 file_hangle_get_name_string()

```
character(len=:)  function, allocatable file_io::file_hangle_get_name_string (
            class(file_handle), intent(inout) this )  [private]
```

Get the file name associated with the file handle. If the file name is (yet) undefined, the latest operation success flag (see file_io::is_success()) is FALSE. **Example:**

```
print *, data_file%get_name()
```

Returns

the name of the file.

Definition at line 260 of file m_fileio.f90.

8.2.4.3 file_object_get_associated_unit()

```
integer function file_io::file_object_get_associated_unit (
            class(file_handle), intent(inout) this )  [private]
```

A Low level function to get the Fortran unit number associated with the file handle object.

Note

This function is useful only for diagnostics because Fortran unit is treated automatically and transparently in all the routines of this module. Of course, it could also be useful for low-level code. **Example:**

```
print *, data_file%get_unit()
```

Returns

the Furtran file unit.

Definition at line 278 of file m_fileio.f90.

8.2.4.4 file_object_format_is_csv()

```
logical function file_io::file_object_format_is_csv (
            class(file_handle), intent(in) this )  [private]
```

Check if the file format is CSV.

Returns

TRUE if the file format is CSV, FALSE otherwise.

Definition at line 288 of file m_fileio.f90.

8.2.4.5 file_object_format_is_txt()

```
logical function file_io::file_object_format_is_txt (
            class(file_handle), intent(in) this )  [private]
```

Check if the file format is CSV.

Returns

TRUE if the file format is TXT, FALSE otherwise.

Definition at line 301 of file m_fileio.f90.

8.2.4.6 csv_open_write_this()

```
subroutine file_io::csv_open_write_this (
            class(file_handle), intent(inout) this,
            character(len=*), intent(in) name,
            integer, intent(in), optional format )  [private]
```

This is an object oriented wrapper for `CSV_OPEN_WRITE()`. For details see `CSV_OPEN_WRITE`.

Note

This procedure also automatically and transparently assigns the Fortran unit. **Example:**

```
call data_file%open_write( "file_001.csv" )
```

Parameters

in	*name*	name the name of the file.
in	*format*	format optional data format type of the file: • FORMAT_CSV (default) • FORMAT_TXT

8.2.4.6.1 Implementation notes Set name from the mandatory `name` argument.
Default format is `FORMAT_CSV`.
Providing a non-supported format results in FALSE status flag.
Open the file for writing physically.
Definition at line 321 of file m_fileio.f90.

8.2.4.7 csv_close_this()

```
subroutine file_io::csv_close_this (
            class(file_handle), intent(inout) this )  [private]
```

This is an object oriented wrapper for `CSV_CLOSE()`. For details see `CSV_CLOSE`.

Note

This procedure also automatically ans transparently assigns the Fortran unit. **Example:**
```
call data_file%close()
```

Definition at line 373 of file m_fileio.f90.

8.2.4.8 csv_header_line_write_this()

```
subroutine file_io::csv_header_line_write_this (
            class(file_handle), intent(inout) this,
            character(len=*), intent(in), optional header )  [private]
```

This is an object oriented wrapper for `CSV_HEADER_WRITE()`. See `CSV_HEADER_WRITE` for details.

Note

File header is optional in CSV files and is not used in most cases. **Example:**
```
call data_file%header_write("Agent data at start of the simulation")
```

Parameters

in	*header*	header is the optional header line for the CSV file. If header is absent, it is automatically generated from the file name.

Definition at line 387 of file m_fileio.f90.

8.2.4.9 csv_record_string_write_this()

```
subroutine file_io::csv_record_string_write_this (
            class(file_handle), intent(inout) this,
            character(len=*), intent(in) csv_record )  [private]
```

Physically write a single string CSV data record to the file. See `CSV_RECORD_WRITE` **Example:**
```
call data_file%record_write( record_string )
```

Parameters

in	*csv_record*	csv_record character string that keeps the whole record (row) of the CSV data spreadsheet.

Definition at line 408 of file m_fileio.f90.

8.2.5 Variable Documentation

8.2.5.1 undefined

```
@, public file_io::undefined
```

Definition at line 121 of file m_fileio.f90.

8.2.5.2 format_csv

```
@, public file_io::format_csv
```

Definition at line 121 of file m_fileio.f90.

8.2.5.3 format_txt

```
@, public file_io::format_txt
```

Definition at line 121 of file m_fileio.f90.

8.3 the_behaviour Module Reference

Definition of high level behavioural architecture.

Data Types

- type behaviour_base

  Root behaviour abstract type. Several different discrete behaviours encompass the behavioural repertoire of the agent. This is the base root type from which all other behaviours are obtained by inheritance/extension.
- interface behaviour_init_root

  *Abstract interface for the deferred **init** function that has to be overriden by each object that extends the basic behavioural component class.*
- type move

  Movement is an umbrella abstract type linked with spatial movement.
- interface move_init_root

  *Abstract interface for the deferred **init** function that has to be overriden by each object that extends the basic behavioural component class.*
- type eat_food

  ***Eat food** is consuming food item(s) perceived.*
- type reproduce

  Reproduce is do a single reproduction.
- type walk_random

  ***Walk_random** is a single step of a Gaussian random walk.*
- type freeze

  ***Freeze** is stop any locomotion completely.*
- type escape_dart

  ***Escape dart** is a very fast long distance movement, normally in response to a direct predation threat.*
- type approach

  ***Approach an arbitrary spatial object** is a directed movement to an arbitrary the_environment::spatial class target object.*
- type approach_conspec

  ***Approach conspecifics** is directed movement towards a conspecific.*
- type migrate

  ***Migrate** is move quickly directing to the other habitat*
- type go_down_depth

  Go down dive deeper.
- type go_up_depth

  Go up raise to a smaller depth. TODO: abstract type linking both Up and Down.
- type debug_base

  This is a test fake behaviour unit that is used only for debugging. It cannot be "execute"'d, but the expectancy can be calculated (normally in the debug mode).
- type behaviour

  The behaviour of the agent is defined by the the_behaviour::behaviour class. This class defines the behavioural repertoire of the agent. Each of the components of the behavioural repertoire (behaviour object) is defined as a separate independent class with its own self parameter. However, the agent which performs the behaviour (the actor agent) is included as the first non-self parameter into the behaviour component methods.
- type architecture_neuro

  This type is an "umbrella" for all the lower-level classes.

Functions/Subroutines

- pure subroutine behaviour_root_attention_weights_transfer (this, this_agent)

  *Transfer attention weights from the actor agent to the behaviour's GOS expectancy object. At this stage, attention weights for **this** behaviour's expectancy motivational state components are copied from the actor agent's (`this_agent`) main motivational components' attention weights.*
- elemental real(srp) function behaviour_root_gos_expectation (this)

  Accessor get-function for the final expected GOS arousal from this behaviour. All calculations for are done in `expectancies_calculate` for the specific behaviour unit.
- elemental logical function behaviour_root_get_is_executed (this)

  Get the execution status of the behaviour unit. If TRUE, the unit is currently active and is being executed. This is the "getter" for the_behaviour::behaviour_base::is_active.
- elemental subroutine eat_food_item_init_zero (this)

  *Initialise the **eat food item** behaviour component to a zero state.*
- elemental subroutine walk_random_init_zero (this)

  *Initialise the **walk_random** behaviour component to a zero state.*
- elemental subroutine freeze_init_zero (this)

  *Initialise the **freeze** behaviour component to a zero state. Freeze is a special type of move to a zero distance / zero speed.*
- subroutine freeze_do_this (this, this_agent)

  Do freeze by `this_agent` (the actor agent). Subjective assessment of the motivational value for this is based on the number of food items, conspecifics and predators in the perception object.
- subroutine freeze_motivations_expect (this, this_agent, time_step_model, rescale_max_motivation)

  `the_behaviour::freeze::motivations_expect()` (re)calculates motivations from fake expected perceptions following from the procedure `freeze::do_this()` => `the_behaviour::freeze_do_this()`.
- subroutine freeze_do_execute (this, this_agent)

  Execute this behaviour component "freeze" by `this_agent` agent.
- elemental subroutine escape_dart_init_zero (this)

  *Initialise the **escape dart** behaviour component to a zero state. Dart is a quick high speed active escape.*
- subroutine escape_dart_do_this (this, this_agent, predator_object, dist_is_stochastic, time_step_model)

  Do active escape dart by `this_agent` (the actor agent). Subjective assessment of the motivational value for this is based on the distance of escape (in turn, dependent on the visibility of the predator).
- subroutine escape_dart_motivations_expect (this, this_agent, predator_object, time_step_model, rescale_max_motivation)

  `escape_dart::motivations_expect()` is a subroutine (re)calculating motivations from fake expected perceptions following from the procedure `escape_dart::do_this()` => `the_behaviour::escape_dart_do_this()`.
- subroutine escape_dart_do_execute (this, this_agent, predator_object, environment_limits)

  Execute this behaviour component "escape" by `this_agent` agent.
- elemental subroutine approach_spatial_object_init_zero (this)

  *Initialise the **approach** behaviour component to a zero state. Approach is a generic type but not abstract.*
- subroutine approach_do_this (this, this_agent, target_object, target_offset, predict_window_food, time_step_model)

  The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here `APPROACH`).
- subroutine approach_motivations_expect (this, this_agent, target_object, target_offset, time_step_model, rescale_max_motivation)

  `the_behaviour::approach::expectancies_calculate()` (re)calculates motivations from fake expected perceptions following from the procedure `approach::do_this()` => `the_behaviour::approach_do_this()`.
- subroutine approach_do_execute (this, this_agent, target_object, is_random, target_offset, environment_limits)

  Execute this behaviour component "approach" by `this_agent` agent.
- elemental subroutine approach_conspecifics_init_zero (this)

  *Initialise the **approach conspecific** behaviour to a zero state. Approach conspecific is a special extension of the generic `APPROACH` behaviour.*

- subroutine approach_conspecifics_do_this (this, this_agent, target_object, target_offset, predict_window_↩ food, time_step_model)

  The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here `APPROACH_CONSPEC`).
- subroutine approach_conspecifics_motivations_expect (this, this_agent, target_object, target_offset, time↩ _step_model, rescale_max_motivation)

  `the_behaviour::approach_conspec::expectancies_calculate()` *(re)calculates motivations from fake expected perceptions following from the procedure the_behaviour::approach_conspec::do_this().*
- elemental subroutine migrate_init_zero (this)

  *Initialise the **migrate** behaviour component to a zero state.*
- subroutine migrate_do_this (this, this_agent, target_env, predict_window_food, predict_window_consp, predict_window_pred, time_step_model)

  The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here `MIGRATE`).
- subroutine migrate_motivations_expect (this, this_agent, target_env, predict_window_food, predict_↩ window_consp, predict_window_pred, time_step_model, rescale_max_motivation)

  `the_behaviour::migrate::expectancies_calculate()` *(re)calculates motivations from fake expected perceptions following from the procedure* `migrate::do_this()`.
- subroutine migrate_do_execute (this, this_agent, target_env)

  Execute this behaviour component "migrate" by `this_agent` agent.
- pure real(srp) function hope (baseline, memory_old, memory_new, zero_hope, maximum_hope, raw_grid_x, raw_grid_y)

  The hope function for the assessment of expectancy for a completely novel stimulus or environment for which local information is absent.
- elemental real(srp) function depth_walk_default (length, walk_factor)

  Calculate the default upward and downward walk step size. This function is called from the_behaviour::go_down_do_this() and the_behaviour::go_down_motivations_expect() if the upwards or downwards walk size is not provided explicitly.
- elemental subroutine go_down_depth_init_zero (this)

  *Initialise the **go down to a deeper spatial layer** behaviour component to a zero state.*
- subroutine go_down_do_this (this, this_agent, max_depth, depth_walk, predict_window_food, time_step_↩ model)

  Do go down by `this_agent` (the actor agent). Subjective assessment of the motivational value for this is based on the number of food items, conspecifics and predators at the layers below the `this_agent` actor agent.
- subroutine go_down_motivations_expect (this, this_agent, depth_walk, max_depth, environments, time_↩ step_model, rescale_max_motivation)

  `go_down_depth::motivations_expect()` *is a subroutine (re)calculating motivations from fake expected perceptions following from the procedure* `go_down_depth::do_this()` => `the_behaviour::go_down_do_this()`.
- subroutine go_down_do_execute (this, this_agent, max_depth, environments, depth_walk)

  Execute this behaviour component "go down" by `this_agent` agent.
- elemental subroutine go_up_depth_init_zero (this)

  *Initialise the **go up to a shallower spatial layer** behaviour component to a zero state.*
- subroutine go_up_do_this (this, this_agent, min_depth, depth_walk, predict_window_food, time_step_model)

  Do go up by `this_agent` (the actor agent). Subjective assessment of the motivational value for this is based on the number of food items, conspecifics and predators at the layers below the `this_agent` actor agent.
- subroutine go_up_motivations_expect (this, this_agent, depth_walk, min_depth, environments, time_step_↩ model, rescale_max_motivation)

  `go_up_depth::motivations_expect()` *is a subroutine (re)calculating motivations from fake expected perceptions following from the procedure* `go_up_depth::do_this()` => `the_behaviour::go_up_do_this()`.
- subroutine go_up_do_execute (this, this_agent, min_depth, environments, depth_walk)

  Execute this behaviour component "go up" by `this_agent` agent towards.
- elemental subroutine debug_base_init_zero (this)

  *Initialise the **fake debug behaviour** behaviour component to a zero state.*

- subroutine debug_base_motivations_expect (this, this_agent, time_step_model, rescale_max_motivation)

  *`the_behaviour::debug_base::motivations_expect()` is a subroutine (re)calculating motivations from fake expected perceptions for the **fake debug behaviour**.*
- subroutine eat_food_item_do_this (this, this_agent, food_item_eaten, time_step_model, distance_food_item, capture_prob, is_captured)

  *Eat a food item defined by the object `food_item_eaten`. The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here `the_behaviour::eat_food`). So, here the result of this procedure is assessment of the stomach content increment and body mass increment that would result from eating the **this** food item by the **this_agent**. The **main output** from this **do** procedure is the `this` behavioural unit, namely two of its internal data components:*
- subroutine eat_food_item_motivations_expect (this, this_agent, food_item_eaten, time_step_model, distance_food_item, capture_prob, rescale_max_motivation)

  `eat_food::motivations_expect()` is a subroutine (re)calculating motivations from fake expected perceptions following from the procedure `eat_food::do_this()` => `the_behaviour::eat_food_item_do_this()`.
- subroutine eat_food_item_do_execute (this, this_agent, food_item_eaten, food_resource_real, eat_is_success)

  Execute this behaviour component "eat food item" by `this_agent` agent towards the `food_item_eaten`.
- elemental subroutine reproduce_init_zero (this)

  Initialise reproduce behaviour object.
- integer function maximum_n_reproductions (this)

  Calculate the maximum number of possible reproductions for this agent. It is assumed that a male can potentially fertilise several females that are within its perception object (in proximity) during a single reproduction event. For females, this number if always one.
- subroutine reproduce_do_this (this, this_agent, p_reproduction, is_reproduce)

  *Do reproduce by `this_agent` (the actor agent) given the specific probability of successful reproduction. The probability of reproduction depends on the number of agents of the same and of the opposite sex within the visual range of the this agent weighted by the difference in the body mass between the actor agent and the average body mass of the other same-sex agents. The **main output** from this **do** procedure is the `this` behavioural unit object, namely its two components:*
- subroutine reproduce_motivations_expect (this, this_agent, time_step_model, reprod_prob, non_stochastic, rescale_max_motivation)

  `reproduce::motivations_expect()` is a subroutine (re)calculating motivations from fake expected perceptions following from `reproduce::do_this()` => `the_behaviour::reproduce_do_this()` procedure.
- subroutine reproduce_do_execute (this, this_agent)

  Execute this behaviour component "reproduce" by the `this_agent` agent.
- subroutine walk_random_do_this (this, this_agent, distance, distance_cv, predict_window_pred, predict_window_food, time_step_model)

  The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here `WALK_RANDOM`).
- subroutine walk_random_motivations_expect (this, this_agent, distance, distance_cv, predict_window_pred, predict_window_food, time_step_model, rescale_max_motivation)

  `the_behaviour::walk_random::expectancies_calculate()` (re)calculates motivations from fake expected perceptions following from the procedure `walk_random::do_this()` => `the_behaviour::walk_random_do_th`
- subroutine walk_random_do_execute (this, this_agent, step_dist, step_cv, environment_limits)

  Execute this behaviour component "random walk" by `this_agent` agent.
- elemental subroutine, private behaviour_whole_agent_init (this)

  Initialise the behaviour components of the agent, the the_behaviour::behaviour class.
- elemental subroutine behaviour_whole_agent_deactivate (this)

  Deactivate all behaviour units that compose the behaviour repertoire of the agent.
- elemental character(len=label_length) function behaviour_get_behaviour_label_executing (this)

  Obtain the label of the currently executing behaviour for the `this` agent.
- integer function behaviour_select_conspecific (this, rescale_max_motivation)

*Select the optimal conspecific among (possibly) several ones that are available in the **perception object** of the agent.*

- integer function behaviour_select_conspecific_nearest (this)

  Select the nearest conspecific among (possibly) several ones that are available in the perception object. Note that conspecifics are sorted by distance within the perception object. Thus, this procedure just selects the first conspecific.
- integer function behaviour_select_food_item (this, rescale_max_motivation)

  *Select the optimal food item among (possibly) several ones that are available in the **perception object** of the agent.*
- integer function behaviour_select_food_item_nearest (this)

  *Select the nearest food item among (possibly) several ones that are available in the perception object. This is a specific and **most simplistic** version of the `behaviour_select_food_item` function: select the nearest food item available in the agent's perception object. Because the food items are sorted within the perception object just select the first item.*
- subroutine behaviour_do_eat_food_item (this, number_in_seen, food_resource_real)

  Eat a specific food item that are found in the perception object.
- subroutine behaviour_do_reproduce (this)

  Reproduce based on the `this` agent's current state.
- subroutine behaviour_do_walk (this, distance, distance_cv)

  Perform a random Gaussian walk to a specific average distance with certain variance (set by the CV).
- subroutine behaviour_do_freeze (this)

  Perform (execute) the the_behaviour::freeze behaviour.
- subroutine behaviour_do_escape_dart (this, predator_object)

  Perform (execute) the the_behaviour::escape_dart behaviour.
- subroutine behaviour_do_approach (this, target_object, is_random, target_offset)

  Approach a specific the_environment::spatial class target, i.e. execute the the_behaviour::approach behaviour. The target is either a conspecific from the perception (the_neurobio::conspec_percept_comp class) or any arbitrary the_environment::spatial class object.
- subroutine behaviour_do_migrate (this, target_env)

  Perform (execute) the the_behaviour::migrate (migration) behaviour.
- logical function behaviour_try_migrate_random (this, target_env, max_dist, prob)

  Perform a simplistic random migration. If the agent is within a specific distance to the target environment, it emigrates there with a specific fixed probability.
- subroutine behaviour_do_go_down (this, depth_walk)

  Perform (execute) the the_behaviour::go_down_depth (go down) behaviour.
- subroutine behaviour_do_go_up (this, depth_walk)

  Perform (execute) the the_behaviour::go_up_depth (go up) behaviour.
- elemental subroutine behaviour_cleanup_history (this)

  Cleanup the behaviour history stack for the agent. All values are empty.
- subroutine behaviour_select_optimal (this, rescale_max_motivation, food_resource_real)

  *Select and **execute** the optimal behaviour, i.e. the behaviour which minimizes the expected GOS arousal.*
- subroutine behaviour_select_fixed_from_gos (this, rescale_max_motivation, food_resource_real)

  *Select and **execute** behaviour based on the current global organismic state. This procedure is significantly different from the_behaviour::behaviour_select_optimal() in that the behaviour that is executed is not based on optimisation of the expected GOS. Rather, the current GOS fully determines which behaviour unit is executed. Such a rigid link necessarily limits the range of behaviours that could be executed.*
- elemental subroutine, private neurobio_init_components (this)

  Initialise neuro-biological architecture.

Variables

- character(len= $*$), parameter, private modname = "(THE_BEHAVIOUR)"

8.3.1 Detailed Description

Definition of high level behavioural architecture.

8.3.2 THE_BEHAVIOUR module

This module defines the behavioural architecture of the agent, extending the starting neutobiology defined in the_neurobio. Various behavioural actions are implemented that form the behavioural repertoire of the agent.

8.3.3 Function/Subroutine Documentation

8.3.3.1 behaviour_root_attention_weights_transfer()

```
pure subroutine the_behaviour::behaviour_root_attention_weights_transfer (
            class(behaviour_base), intent(inout) this,
            class(appraisal), intent(in) this_agent )
```

Transfer attention weights from the actor agent to the behaviour's GOS expectancy object. At this stage, attention weights for **this** behaviour's expectancy motivational state components are copied from the actor agent's (this↩_agent) main motivational components' attention weights.

Note

The associate construct makes it easier to write all possible combinations, so there is little need to implement motivation-state specific attention transfer functions separately. Here in the below associate constructs EX is the this **expectancy** class root and AG is the **actor agent** class root.

Attention transfer routine cannot be conveniently placed into the STATE_MOTIVATION_BASE because specific motivation states (hunger,...) are still unavailable at this level, but we are intended to get access to specific motivational state of the actor agent. The state_motivation_attention_weights↩_transfer procedure in STATE_MOTIVATION_BASE class just implements attention weights transfer across two **motivation state** root class objects. Even so, we still would need this function here calling specific motivation object-bound versions. However, this is more complicated than just a single subroutine as implemented here for the BEHAVIOUR_BASE. Anyway, we only really copy attention weights for all motivation states in a single batch here and never need it elsewhere.

Note

We have to include all the motivation state components that are found in the MOTIVATION class, hunger, fear_defence etc.

Transfer attention weights for **hunger**.

Note

The STATE_MOTIVATION_BASE bound procedure that implements this attention transfer is: call thisexpectancyhungerattention_copy(& this_agentmotivationshunger)

Transfer attention weights for **fear_defence**.

Note

The STATE_MOTIVATION_BASE bound procedure that implements this attention transfer is: call thisexpectancyfear_defenceattention_copy(& this_agentmotivationsfear_↩defence)

Transfer attention weights for **reproduction**.

Note

The STATE_MOTIVATION_BASE bound procedure that implements this attention transfer is: call thisexpectancyreproductionattention_copy(& this_agentmotivationsreproduction)

Definition at line 669 of file m_behav.f90.

8.3.3.2 behaviour_root_gos_expectation()

```
elemental real(srp) function the_behaviour::behaviour_root_gos_expectation (
            class(behaviour_base), intent(in) this )
```

Accessor get-function for the final expected GOS arousal from this behaviour. All calculations for are done in `expectancies_calculate` for the specific behaviour unit.

Parameters

in	*this*	this self.

Returns

Expected GOS arousal level if **this** behaviour is executed.

Definition at line 745 of file m_behav.f90.

8.3.3.3 behaviour_root_get_is_executed()

```
elemental logical function the_behaviour::behaviour_root_get_is_executed (
            class(behaviour_base), intent(in) this )
```

Get the execution status of the behaviour unit. If TRUE, the unit is currently active and is being executed. This is the "getter" for the_behaviour::behaviour_base::is_active.

Returns

TRUE, the behaviour unit is currently active and is being executed; FALSE otherwise.

Definition at line 758 of file m_behav.f90.

8.3.3.4 eat_food_item_init_zero()

```
elemental subroutine the_behaviour::eat_food_item_init_zero (
            class(eat_food), intent(inout) this )
```

Initialise the **eat food item** behaviour component to a zero state.
First init components from the base root class `the_behaviour::behaviour_base`: Mandatory label component that should be read-only.
The execution status is always FALSE, can be reset to TRUE only when the behaviour unit is called to execution.
And the *expectancy* type components.
And init the expected arousal data component.
Second, init components of this specific behaviour (`EAT_FOOD`) component extended class.

Note

Note that we initialise increments to 0.0, not MISSING as increments will be later added. And several items can be added consecutively.

Definition at line 774 of file m_behav.f90.

8.3.3.5 walk_random_init_zero()

```
elemental subroutine the_behaviour::walk_random_init_zero (
            class(walk_random), intent(inout) this )
```

Initialise the **walk_random** behaviour component to a zero state.
First, initialise components from the base root class `the_behaviour::behaviour_base`. Mandatory label component that should be read-only.
The execution status is always FALSE, can be reset to TRUE only when the behaviour unit is called to execution.
And the *expectancy* components.
Abstract `MOVE` component.

Second, init components of this specific behaviour (WALK_RANDOM).
Definition at line 802 of file m_behav.f90.

8.3.3.6 freeze_init_zero()

```
elemental subroutine the_behaviour::freeze_init_zero (
            class(freeze), intent(inout) this )
```

Initialise the **freeze** behaviour component to a zero state. Freeze is a special type of move to a zero distance / zero speed.
First init components from the base root class the_behaviour::behaviour_base. Mandatory label component that should be read-only.
The execution status is always FALSE, can be reset to TRUE only when the behaviour unit is called to execution.
And the *expectancy* components.
Abstract MOVE component.
Second, init components of this specific behaviour (FREEZE).
Definition at line 833 of file m_behav.f90.

8.3.3.7 freeze_do_this()

```
subroutine the_behaviour::freeze_do_this (
            class(freeze), intent(inout) this,
            class(appraisal), intent(in) this_agent )
```

Do freeze by this_agent (the actor agent). Subjective assessment of the motivational value for this is based on the number of food items, conspecifics and predators in the perception object.

Parameters

in,out	*this*	[inout] this the object itself.
in	*this_agent*	this_agent is the actor agent which goes down.

8.3.3.7.1 Implementation details The expected food gain for freezing is zero as immobile agent does not eat.
Calculate the expected direct risk of predation that is based on the distance to the nearest predator. However, a version of the the_neurobio::perception::risk_pred() procedure for freezing/immobile agent is used here.
Calculate the expected predation risk for the immobile agent. It is assumed that predators that are roaming nearby cannot easily detect an immobile/freezing agent as long as it does not move (freezing here has significant similarity with sheltering). Therefore, the expectancy is based on a (subjective) **zero** count of the number of predators in the agent's perception object and normal risk component based on the predators in the memory stack. The calculation is done by the standard the_neurobio::predation_risk_backend() function. Thus, the resulting general risk is calculated as:

$$R = 0 + r_{id} \cdot (1 - \omega),$$

where r_{id}) is the average number of predators in the latest memory stack and ω is the weighting factor for the actual number of predators (that is zero in this case).
Definition at line 862 of file m_behav.f90.

8.3.3.8 freeze_motivations_expect()

```
subroutine the_behaviour::freeze_motivations_expect (
            class(freeze), intent(inout) this,
            class(appraisal), intent(in) this_agent,
            integer, intent(in), optional time_step_model,
            real(srp), intent(in), optional rescale_max_motivation )
```

the_behaviour::freeze::motivations_expect() (re)calculates motivations from fake expected perceptions following from the procedure freeze::do_this() => the_behaviour::freeze_do_this().

Parameters

`in,out`	*this*	[inout] this the object itself.
`in`	*this_agent*	this_agent is the actor agent which does freezing.
`in`	*time_step_model*	[in] time_step_model optional time step of the model, **overrides** the value calculated from the spatial data.
`in`	*rescale_max_motivation*	rescale_max_motivation optional maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

8.3.3.8.1 Notable local variables

8.3.3.8.1.1 Perception overrides

- **expect_pred_dir** is the expected direct predation risk; it is zero.

expect_predator is the expected general predation risk, that is based on a weighting of the current predation and predation risk from the memory stack.

- **expect_stomach** is the expected stomach contents as a consequence of freezing. Note that there is no food consumption while freezing.
- **expect_bodymass** is the expected body mass as a consequence of freezing. Notably, it subtracts a small living cost component.
- **expect_energy** is the expected energy reserves as a consequence of the freezing. Calculated from the body mass and weight.

8.3.3.8.1.2 Checks and preparations Check optional time step parameter. If not provided, use global parameter value from commondata::global_time_step_model_current.

8.3.3.8.1.3 Call do_this As the first step, we use the **do**-procedure `freeze::do_this()` => `the_behaviour::freeze_do_this()` to perform the behaviour desired without changing either the agent or its environment, obtaining the **subjective** values of the `this` behaviour components that later feed into the motivation **expectancy** functions:

- `perception_override_pred_dir`
- `perception_override_predator`
- `perception_override_stomach`
- `perception_override_bodymass`
- `perception_override_energy`

8.3.3.8.1.4 Calculate expected (fake) perceptions First, calculate the expected stomach contents, body mass and energy reserves out of the fixed zero food gain that is returned from the `do_this` procedure.

- Obtain the agent's current stomach contents.

Calculate the expected stomach content, which is decremented by the expected digestion value (the_body::stomach_emptify_backend

- Calculate the expected body mass of the agent as a consequence of freezing. The body mass is decremented by a small value of the living cost (the_body::body_mass_calculate_cost_living_step()).
- The expected energy reserves are calculated from the fake perceptions of the body mass and the current length (it does not change as food intake is zero in case of freezing) using the_body::energy_reserve() function.

Second, transfer the predation risk expectancies from the freezing class object to the dedicated override perception variables (their final values are calculated in `do_this`).

8.3.3.8.1.5 Calculate motivation expectancies The next step is to calculate the motivational expectancies using the fake perceptions to override the default (actual agent's) values. At this stage, first, calculate motivation values resulting from the behaviour done (`freeze::do_this()`) at the previous steps: what would be the motivation values *if* the agent does perform FREEZE? Technically, this is done by calling the **neuronal response function**, `percept_components_motiv::motivation_components()` method, for each of the motivational states with `perception_override_` dummy parameters overriding the default values. Here is the list of the fake overriding perceptions for the FREEZE behaviour:

- `perception_override_pred_dir`
- `perception_override_predator`
- `perception_override_stomach`
- `perception_override_bodymass`
- `perception_override_energy`

Real agent perception components are now substituted by the *fake* values resulting from executing this behaviour (`reproduce::do_this()` => `the_behaviour::reproduce_do_this()` method). This is repeated for all the motivations: *hunger*, *fear state* etc. These optional **override parameters** are substituted by the "fake" values.

8.3.3.8.1.6 Calculate primary and final motivations Next, from the perceptual components calculated at the previous step we can obtain the **primary** and **final motivation** values by weighed summing.
Here we can use global maximum motivation across all behaviours and perceptual components if it is provided, for rescaling.
Or can rescale using local maximum value for this behaviour only.
Transfer attention weights from the actor agent `this_agent` to the `this` behaviour component. So, we will now use the updated modulated attention weights of the agent rather than their default parameter values.
So the primary motivation values are calculated.
Primary motivations are logged in the debug mode.
There is **no modulation** at this stage, so the final motivation values are the same as primary motivations.

8.3.3.8.1.7 Calculate motivation expectancies Finally, calculate the finally **expected arousal level for this behaviour**. As in the GOS, the overall arousal is the maximum value among all motivation components.
Log also the final expectancy value in the debug mode.
Now as we know the expected arousal, we can choose the behaviour which would minimise this arousal level.
Definition at line 918 of file m_behav.f90.

8.3.3.9 freeze_do_execute()

```
subroutine the_behaviour::freeze_do_execute (
             class(freeze), intent(inout) this,
             class(appraisal), intent(inout) this_agent )
```

Execute this behaviour component "freeze" by `this_agent` agent.

Parameters

`in,out`	*this_agent*	[in] this_agent is the actor agent which goes down.

8.3.3.9.1 Implementation details

8.3.3.9.1.1 Step 1: do_this As the first step, we use the **do**-procedure `freeze::do_this()` to perform the behaviour desired. As a result, the following values are obtained:

- expected zero food gain

- expected zero direct predation risk
- expected general predation risk, assuming no direct threat.

However, because freezing does not incur any specific behavioural costs and does not change any environmental objects, calling `do_this()` is really **unnecessary**. It is therefore only called in the DEBUG mode to log and check the resulting perception values.

8.3.3.9.1.2 Step 2: Change the agent Freezing results in some small cost, equal to a single piece of the the cost of living. However, it is much smaller than the cost of locomotion. Also, no food can be obtained while freezing but digestion still occurs, so the value of the stomach contents is reduced by a fixed fraction. However, freezing, unlike other behaviour components, does not incur any specific cost or change of the agent. Cost of living and digestion subtractions are updated for every time step for every other behaviours anyway. Therefore, it is **not** done here.

8.3.3.9.1.3 Step 3: Change the environment Freezing does not affect the environmental objects.
Definition at line 1221 of file m_behav.f90.

8.3.3.10 escape_dart_init_zero()

```
elemental subroutine the_behaviour::escape_dart_init_zero (
            class(escape_dart), intent(inout) this )
```

Initialise the **escape dart** behaviour component to a zero state. Dart is a quick high speed active escape.
First init components from the base root class `the_behaviour::behaviour_base`. Mandatory label component that should be read-only.
The execution status is always FALSE, can be reset to TRUE only when the behaviour unit is called to execution.
And the *expectancy* components.
Abstract `MOVE` component.
Second, init components of this specific behaviour (`ESCAPE_DART`).
Definition at line 1266 of file m_behav.f90.

8.3.3.11 escape_dart_do_this()

```
subroutine the_behaviour::escape_dart_do_this (
            class(escape_dart), intent(inout) this,
            class(appraisal), intent(in) this_agent,
            class(spatial), intent(in), optional predator_object,
            logical, intent(in), optional dist_is_stochastic,
            integer, intent(in), optional time_step_model )
```

Do active escape dart by `this_agent` (the actor agent). Subjective assessment of the motivational value for this is based on the distance of escape (in turn, dependent on the visibility of the predator).

Parameters

in	*this_agent*	this_agent is the actor agent which goes down.
in	*predator_object*	predator_object optional predator object, if present, it is assumed the actor agent tries to actively escape from this specific predator.
in	*dist_is_stochastic*	dist_is_stochastic Logical flag, if set to TRUE, the escape distance is stochastic in the expectancy engine; this can define an internal expectation uncertainty.
in	*time_step_model*	time_step_model optional time step of the model, overrides the value calculated from the spatial data.

8.3.3.11.1 Implementation details

8.3.3.11.1.1 Checks and preparations Check optional time step parameter. If unset, use global commondata::global_time_step_model_current.

8.3.3.11.1.2 Calculate expected food gain The expected food gain for active escape is zero as the agent cannot eat at this time.

8.3.3.11.1.3 Calculate cost of fast escape movement First, calculate the distance of escape. The escape distance, in turn, depends on the visibility distance of the predator object: it should exceed this distance, so the actor agent could not see the predator any more.

Visibility range of the predator First, check if the predator object is provided. If the predator object is provided as a dummy parameter, visibility range can be assessed using its size. However, the calculations depend on the exact type of the predator object because it can be the_environment::predator or the_neurobio::spatialobj_percept_comp (in predator perception: the_neurobio::percept_predator) or perhaps even just any extension of the the_environment::spatial class. Fortran `select type` construct is used here.

- If the type of the object is the_environment::predator, then visibility benefits from the object-bound function the_environment::predator::visibility().
- If the object type is the_neurobio::spatialobj_percept_comp (as in perception objects), the visibility is calculated using the object bound function the_neurobio::spatialobj_percept_comp::visibility() with the default object type (`object_area` parameter is not provided), so the object area is calculated for *fish* (see the_neurobio::spatialobj_percept_visibility_visual_range()).
- If the object type is the default class the_environment::spatial, e.g. the_environment::spatial_moving, its size may not be available; the visibility is calculated manually using the_environment::visual_range() function assuming default predator size set by the commondata::predator_body_size parameter.

If the predator object is not provided as a dummy parameter, visibility range is assessed using the default size of the predator commondata::predator_body_size and the ambient illumination at the actor agent's depth.

Exact escape distance Knowing the visibility range of the predator, one can calculate the escape distance. Namely, the escape distance is obtained by multiplying the visibility range by the commondata::escape_dart_distance_default_factor parameter constant.
This constant should normally exceed 1.0. In such a case, the escape distance exceeds the visibility of the predator. However, it should not be too long to avoid extra energetic cost.
If the `dist_is_stochastic` optional parameter is TRUE, the escape distance is stochastic with the mean as above and the coefficient of variation set by the commondata::escape_dart_distance_default_stoch_cv parameter. Stochastic distance can define *uncertainty* in the escape behaviour expectancy.

Cost of movement Knowing the movement distance, it is possible to calculate the cost of movement to this distance using the the_body::condition_cost_swimming_burst() method assuming the swimming is turbulent (so the exponent parameter takes the commondata::swimming_cost_exponent_turbulent value).

8.3.3.11.1.4 Calculate the direct and general risk of predation The expected direct risk of predation is assumed to be commondata::zero.
Accordingly, the general risk of predation taking account both the number of predators in the perception object and the average number of predators in the memory stack is calculated using the the_neurobio::predation_risk_backend() method, assuming there are no predators in perception.
Definition at line 1296 of file m_behav.f90.

8.3.3.12 escape_dart_motivations_expect()

```
subroutine the_behaviour::escape_dart_motivations_expect (
             class(escape_dart), intent(inout) this,
             class(appraisal), intent(in) this_agent,
             class(spatial), intent(in), optional predator_object,
```

```
            integer, intent(in), optional time_step_model,
            real(srp), intent(in), optional rescale_max_motivation )
```

`escape_dart::motivations_expect()` is a subroutine (re)calculating motivations from fake expected perceptions following from the procedure `escape_dart::do_this()` => `the_behaviour::escape_dart_do_this()`

Parameters

in	*this_agent*	this_agent is the actor agent which goes down.
in	*predator_object*	predator_object optional predator object, if present, it is assumed the actor agent tries to actively escape from this specific predator.
in	*time_step_model*	time_step_model optional time step of the model, overrides the value calculated from the spatial data.
in	*rescale_max_motivation*	rescale_max_motivation maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

8.3.3.12.1 Notable local variables

8.3.3.12.1.1 Perception overrides

- **expect_pred_dir** is the expected direct predation risk; it is zero.

expect_predator is the expected general predation risk, that is based on a weighting of the current predation and predation risk from the memory stack.

- **expect_stomach** is the expected stomach contents as a consequence of escape movement. Note that there is no food consumption during escape.
- **expect_bodymass** is the expected body mass as a consequence of the escape movement. Notably, it subtracts the cost of the escape movement.
- **expect_energy** is the expected energy reserves as a consequence of the escape movement. Calculated from the body mass and weight.

8.3.3.12.2 Implementation details

8.3.3.12.2.1 Checks and preparations Check optional time step parameter. If not provided, use global parameter value from commondata::global_time_step_model_current.

8.3.3.12.2.2 Call do_this As the first step, we use the **do**-procedure `go_down_depth::do_this()` => `the_behaviour::go_down_do_this()` to perform the behaviour desired without changing either the agent or its environment, obtaining the **subjective** values of the `this` behaviour components that later feed into the motivation **expectancy** functions:

- `perception_override_pred_dir`
- `perception_override_predator`
- `perception_override_stomach`
- `perception_override_bodymass`
- `perception_override_energy`

8.3.3.12.2.3 Calculate expected (fake) perceptions First, calculate the expected **stomach content**, which is decremented by the expected digestion value (the_body::stomach_emptify_backend()).
Second, calculate the expected **body mass** of the agent as a consequence of the escape movement. The body mass is decremented by the cost of movement to the this%distance and the cost of living (the_body::condition::living_cost()).
The expected **energy reserves** are calculated from the fake perceptions of the body mass and the current length (length does not change as food intake is zero in case of escape) using the the_body::energy_reserve() function.
The expected **direct predation risk** is transferred from the this object (the_behaviour::escape_dart).
The expected **general predation risk** is also transferred from the this object (the_behaviour::escape_dart).

8.3.3.12.2.4 Calculate motivation expectancies The next step is to calculate the motivational expectancies using the fake perceptions to override the default (actual agent's) values. At this stage, first, calculate motivation values resulting from the behaviour done (`the_behaviour::escape_dart::do_this()`) at the previous steps: what would be the motivation values *if* the agent does perform escape? Technically, this is done by calling the **neuronal response function**, `percept_components_motiv::motivation_components()` method, for each of the motivational states with `perception_override_` dummy parameters overriding the default values. Here is the list of the fake overriding perceptions for the `ESCAPE_DART` behaviour:

- `perception_override_pred_dir`
- `perception_override_predator`
- `perception_override_stomach`
- `perception_override_bodymass`
- `perception_override_energy`

Real agent perception components are now substituted by the *fake* values resulting from executing this behaviour (`reproduce::do_this()` => `the_behaviour::reproduce_do_this()` method). This is repeated for all the motivations: *hunger*, *passive avoidance, active avoidance* etc. These optional **override parameters** are substituted by the "fake" values.

8.3.3.12.2.5 Calculate primary and final motivations Next, from the perceptual components calculated at the previous step we can obtain the **primary** and **final motivation** values by weighed summing.
Here we can use global maximum motivation across all behaviours and perceptual components if it is provided, for rescaling.
Or can rescale using local maximum value for this behaviour only.
Transfer attention weights from the actor agent `this_agent` to the `this` behaviour component. So, we will now use the updated modulated attention weights of the agent rather than their default parameter values.
So the primary motivation values are calculated.
Primary motivations are logged in the debug mode.
There is **no modulation** at this stage, so the final motivation values are the same as primary motivations.

8.3.3.12.2.6 Calculate motivation expectancies Finally, calculate the finally **expected arousal level for this behaviour**. As in the GOS, the overall arousal is the maximum value among all motivation components.
Log also the final expectancy value in the debug mode.
Now as we know the expected arousal, we can choose the behaviour which would minimise this arousal level.
Definition at line 1484 of file m_behav.f90.

8.3.3.13 escape_dart_do_execute()

```
subroutine the_behaviour::escape_dart_do_execute (
             class(escape_dart), intent(inout) this,
             class(appraisal), intent(inout) this_agent,
             class(spatial), intent(in), optional predator_object,
             class(environment), intent(in), optional environment_limits )
```

Execute this behaviour component "escape" by `this_agent` agent.

Parameters

`in,out`	*this_agent*	[in] this_agent is the actor agent which goes down.
`in`	*predator_object*	predator_object optional predator object, if present, it is assumed the actor agent tries to actively escape from this specific predator.
`in`	*environment_limits*	environment_limits Limits of the environment area available for the random walk. The moving object cannot get beyond this limit. If this parameter is not provided, the environmental limits are obtained automatically from the global array the_environment::global_habitats_available.

8.3.3.13.1 Implementation details

8.3.3.13.1.1 Step 1: do_this As the first step, we use the **do**-procedure `the_behaviour::escape_dart::do_this()` to perform the behaviour desired. As a result, the following values are obtained:

- **escape distance**;
- expected (zero) food gain;
- expected **stomach contents, body mass and energy reserves**, assuming nonzero cost of movement and lack of feeding while escaping (i.e. zero food gain).
- the estimates of the predation risk are not used here, they only are used in the subjective evaluation phase, when the agent computes expectancies.

In the debug mode, checking and logging the perception values.

8.3.3.13.1.2 Step 2: Change the agent Escape involves a random walk. Thus, the first thing is the agent *displacement*:

- If the predator is present, the agent does a correlated Gaussian random walk the_environment::spatial_moving::corwalk() in a direction roughly opposite to the predator position.
- If the predator is not present, the agent performs a Gaussian walk, to a distance equal to the this%distance data component and the CV set by the parameter commondata::escape_dart_distance_default_stoch_cv.

  Note

  Note that the escape involves a full 3D walk with a single set of distance and CV parameters (i.e. no separate depth walk parameters).

  Escape movement results a *cost* that is defined by the actual distance travelled, the_environment::spatial_moving::way() which is subtracted here. Call `the_body::condition::set_mass()` for this.

  Note

  Note that the_body::condition::cost_swim() calculates the cost of the latest way passed (the_environment::spatial_moving:: if the distance parameter is not provided.

  Additionally, also call the `the_body::condition::set_length()` method to update the body length history stack. However, the value_set parameter here is just the current value. This fake re-setting of the body length is done to keep both mass and length synchronised in their history stack arrays (there is no procedure for only updating history).

After resetting the body mass, update energy reserves of the agent, that depend on both the length and the mass. Check if the agent is starved to death. If yes, the agent can die without going any further.

8.3.3.13.1.3 Step 3: Change the environment Escape movement itself does not affect the environmental objects.
Definition at line 1799 of file m_behav.f90.

8.3.3.14 approach_spatial_object_init_zero()

```
elemental subroutine the_behaviour::approach_spatial_object_init_zero (
            class(approach), intent(inout) this )
```

Initialise the **approach** behaviour component to a zero state. Approach is a generic type but not abstract.
First init components from the base root class the_behaviour::behaviour_base. Mandatory label component that should be read-only.
The execution status is always FALSE, can be reset to TRUE only when the behaviour unit is called to execution.
And the *expectancy* components.
Abstract MOVE component.
Then init components of this specific behaviour component extended class.
Definition at line 1936 of file m_behav.f90.

8.3.3.15 approach_do_this()

```
subroutine the_behaviour::approach_do_this (
            class(approach), intent(inout) this,
            class(appraisal), intent(in) this_agent,
            class(spatial), intent(in) target_object,
            real(srp), intent(in), optional target_offset,
            integer, intent(in), optional predict_window_food,
            integer, intent(in), optional time_step_model )
```

The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here APPROACH).

Parameters

in	*this_agent*	this_agent is the actor agent which eats the food item.
in	*target_object*	target_object is the spatial target object the actor agent is going to approach.
in	*target_offset*	target_offset is an optional offset for the target, so that the target position of the approaching agent does not coincide with the target object. If absent, a default value set by the commondata::approach_offset_default is used.
in	*predict_window_food*	predict_window_food the size of the prediction window, i.e. how many steps back in memory are used to calculate the predicted food gain. This parameter is limited by the maximum commondata::history_size_perception value of the perception memory history size.

Note

This parameter is not used here and is placed only to make derived class subroutine make the same argument list.

Parameters

in	*time_step_model*	time_step_model optional time step of the model, overrides the value calculated from the spatial data. This parameter is not used for this class, it is here only to allow placement of this parameter for higher-order derived classes.

8.3.3.15.1 Implementation details Check the optional parameter for the target offset and set the default one if offset is not provided.

8.3.3.15.1.1 Proximity check The agent approaches the conspecific but to a nonzero distance equal to the target offset value (target_offset). A check is done if the distance between the agent and the conspecific target object is actually smaller than the target offset.

- If so, the agent is already in close proximity to the target and there is no need to do an approach movement.

The approach distance is set to zero.

- The expected cost of approach movement is also zero.
- If the agent is currently at a distance exceeding the target offset, the approach distance towards the target position of the actor agent is calculated as the true distance towards the target conspecific minus the offset value `target_offset`. (Note that whenever the default target offset is set, i.e. an average of the agent and target body sizes, the approach distance depends on the body sizes of both parties; it is also symmetric, i.e. the same if a large agent approaches a small target conspecific or *vice versa*.)
- Check if the distance to the target object exceeds the migration travel maximum value, set as commondata::migrate_dist_max_step body sizes of the agent. This case should never occur if the maximum distance is sufficiently large so that the target object is beyond the agent's visual range. So, nothing is done here except logging a possible error.
- Calculate expected cost of the swimming. The expected cost of swimming in the approach walk step depends on the above approach distance and is calculated using the_body::condition::cost_swim() method assuming *laminar* flow (laminar flow is due to normal relatively slow swimming pattern).

Definition at line 1965 of file m_behav.f90.

8.3.3.16 approach_motivations_expect()

```
subroutine the_behaviour::approach_motivations_expect (
             class(approach), intent(inout) this,
             class(appraisal), intent(in) this_agent,
             class(spatial), intent(in), optional target_object,
             real(srp), intent(in), optional target_offset,
             integer, intent(in), optional time_step_model,
             real(srp), intent(in), optional rescale_max_motivation )
```

`the_behaviour::approach::expectancies_calculate()` (re)calculates motivations from fake expected perceptions following from the procedure `approach::do_this()` => `the_behaviour::approach_do_this()`.

Parameters

in	*this_agent*	this_agent is the actor agent which does approach.
in	*target_object*	target_object is the spatial target object the actor agent is going to approach.
in	*target_offset*	target_offset is an optional offset for the target, so that the target position of the approaching agent does not coincide with the target object. If absent, a default value set by the commondata::approach_offset_default is used.
in	*time_step_model*	time_step_model optional time step of the model, overrides the value calculated from the spatial data. This parameter is not used for this class, it is here only to allow placement of this parameter for higher-order derived classes.
in	*rescale_max_motivation*	rescale_max_motivation optional maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

8.3.3.16.1 Notable local variables

8.3.3.16.1.1 Perception overrides

- **perception_override_bodymass** is the expected body mass as a consequence of the approach movement.

perception_override_energy is the expected energy reserves as a consequence of the escape movement. Calculated from the body mass and weight.

8.3.3.16.2 Implementation details

8.3.3.16.2.1 Call do_this As the first step, we use the **do**-procedure `walk_random::do_this()` => the_behaviour::walk_random_do_this() to perform the behaviour desired without changing either the agent or its environment, obtaining the **subjective** values of the `this` behaviour components that later feed into the motivation **expectancy** functions:

- `perception_override_bodymass`
- `perception_override_energy`

8.3.3.16.2.2 Calculate expected (fake) perceptions **Body mass**: the **body mass** perception override is obtained by subtracting the approach movement cost and the the_body::condition::living_cost() from the current mass. **Energy**: The fake perception values for the energy reserves (`perception_override_energy`) using the `the_body::energy_reserve()` procedure.

8.3.3.16.2.3 Calculate motivation expectancies The next step is to calculate the motivational expectancies using the fake perceptions to override the default (actual agent's) values. At this stage, first, calculate motivation values resulting from the behaviour done (`walk_random::do_this()`) at the previous steps: what would be the motivation values *if* the agent does perform APPROACH? Technically, this is done by calling the **neuronal response function**, `percept_components_motiv::motivation_components()` method, for each of the motivational states with `perception_override_` dummy parameters overriding the default values. Here is the list of the fake overriding perceptions for the `APPROACH` behaviour:

- `perception_override_bodymass`
- `perception_override_energy`

Real agent perception components are now substituted by the *fake* values resulting from executing this behaviour (`approach::do_this()` method). This is repeated for all the motivations: *hunger*, *passive avoidance, fear state* etc. These optional **override parameters** are substituted by the "fake" values.

8.3.3.16.2.4 Calculate primary and final motivations Next, from the perceptual components calculated at the previous step we can obtain the **primary** and **final motivation** values by weighed summing.
Here we can use global maximum motivation across all behaviours and perceptual components if it is provided, for rescaling.
Or can rescale using local maximum value for this behaviour only.
Transfer attention weights from the actor agent `this_agent` to the `this` behaviour component. So, we will now use the updated modulated attention weights of the agent rather than their default parameter values.
So the primary motivation values are calculated.
Primary motivations are logged in the debug mode.
There is **no modulation** at this stage, so the final motivation values are the same as primary motivations.

8.3.3.16.2.5 Calculate motivation expectancies Finally, calculate the finally **expected arousal level for this behaviour**. As in the GOS, the overall arousal is the maximum value among all motivation components.
Log also the final expectancy value in the debug mode.
Now as we know the expected arousal, we can choose the behaviour which would minimise this arousal level.
Definition at line 2072 of file m_behav.f90.

8.3.3.17 approach_do_execute()

```
subroutine the_behaviour::approach_do_execute (
             class(approach), intent(inout) this,
             class(appraisal), intent(inout) this_agent,
             class(spatial), intent(in) target_object,
             logical, intent(in), optional is_random,
             real(srp), intent(in), optional target_offset,
             class(environment), intent(in), optional environment_limits )
```

Execute this behaviour component "approach" by `this_agent` agent.

Parameters

`in,out`	*this_agent*	[in] this_agent is the actor agent which eats the food item.
`in`	*target_object*	target_object is the spatial target object the actor agent is going to approach.
`in`	*is_random*	is_random indicator flag for random correlated walk. If present and is TRUE, the agent approaches to the `target_object` in form of random correlated walk (see the_environment::spatial_moving::corwalk()), otherwise directly.
`in`	*target_offset*	target_offset is an optional offset for the target, so that the target position of the approaching agent does not coincide with the target object. If absent, a default value set by the commondata::approach_offset_default is used. For the the_behaviour::approach_conspec, the default value is as an average of the agent and target conspecific body lengths.
`in`	*environment_limits*	environment_limits Limits of the environment area available for the random walk. The moving object cannot get beyond this limit.

8.3.3.17.1 Implementation details

8.3.3.17.1.1 Checks and preparations First, check the optional parameters

- random walk flag: `is_random`; if the parameter is not provided, the default value FALSE is set so that the agent does a direct approach towards the target object leaving the target offset distance.

target offset: `target_offset`. Note that setting the default value for the target offset involves calling the `select type` construct. Therefore, the default offset for a simple the_behaviour::approach behaviour is equal to the fixed commondata::approach_offset_default value whereas for the the_behaviour::approach_conspec, it is set as an average of the agent and target conspecific body lengths.
Second, copy the spatial location of the target `target_object` to a temporary spatial object `target_object_tmp` to avoid multiple calling the the_environment::spatial::position() method.

Note

This is needed because the `target_object` is **class** and getting location can be only done through the `location` method.

8.3.3.17.1.2 Step 1: do_this First, we use the intent-in **do**-procedure the_behaviour::approach::do_this() to perform the behaviour desired. Here it calculates the distance towards the target object also taking account of the offset parameter.
Also check here if the approach distance exceeds the limit set by the commondata::migrate_dist_max_step parameter. If it does exceed, the agent will move towards the target object, but the distance is reduced according to the limit.

8.3.3.17.1.3 Step 2: Change the agent

Relocate towards the target object Relocate to the target object can be either a correlated random walk in the target direction or direct movement to the target.

- In the former case, the environmental limits can be either provided by the `environment_limits` parameter or obtained automatically from the global array the_environment::global_habitats_available.
  - If the approach distance is less then commondata::zero (i.e. the target object is already at a distance smaller than target offset), the correlated random walk step is set to the target offset.
- If correlated random walk is not enabled (`is_random` parameter is FALSE), the agent goes *directly* towards the target. It actually relocates to a spatial position with the the target offset. The new position of the agent is defined by the the_environment::offset_dist() function subtracting the value of the offset.

– However, if the approach distance is less than commondata::zero, (i.e. the agent is already in proximity of the target object, at a distance smaller than the target offset), the agent "moves" to its *current* position, i.e. no real relocation is done. This situation is logged in the DEBUG mode.

Process the cost of movement

- Reset the body mass of the actor agent subtracting the actual cost of moving that is automatically calculated in the call to the_body::condition::cost_swim(). The the_body::condition::set_mass() method is used here to adjust the mass.

Additionally, also call the `the_body::condition::set_length()` method to update the body length history stack. However, the value_set parameter here is just the current value. This fake re-setting of the body length is done to keep both mass and length synchronised in their history stack arrays (there is no procedure for only updating history).

- After resetting the body mass, update energy reserves of the agent, that depend on both the length and the mass.

Finally, check if the agent is starved to death. If yes, the agent can die without going any further.

8.3.3.17.1.4 Step 3: Change the environment Approach does not affect the environmental objects.
Definition at line 2345 of file m_behav.f90.

8.3.3.18 approach_conspecifics_init_zero()

```
elemental subroutine the_behaviour::approach_conspecifics_init_zero (
            class(approach_conspec), intent(inout) this )
```

Initialise the **approach conspecific** behaviour to a zero state. Approach conspecific is a special extension of the generic APPROACH behaviour.
First init components from the base root class `the_behaviour::behaviour_base`. Mandatory label component that should be read-only.
The execution status is always FALSE, can be reset to TRUE only when the behaviour unit is called to execution.
And the *expectancy* components.
Abstract MOVE component.
Component of APPROACH class. Then init components of this specific behaviour component extended class.
This class, APPROACH_CONSPEC, initialisations.
Definition at line 2587 of file m_behav.f90.

8.3.3.19 approach_conspecifics_do_this()

```
subroutine the_behaviour::approach_conspecifics_do_this (
            class(approach_conspec), intent(inout) this,
            class(appraisal), intent(in) this_agent,
            class(spatial), intent(in) target_object,
            real(srp), intent(in), optional target_offset,
            integer, intent(in), optional predict_window_food,
            integer, intent(in), optional time_step_model )
```

The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here APPROACH_CONSPEC).

Parameters

in	*this_agent*	this_agent is the actor agent which approaches.
in	*target_object*	target_object is the target conspecific the actor agent is going to approach.

Parameters

in	*target_offset*	target_offset is an optional offset for the target, so that the target position of the approaching agent does not coincide with the target object. If absent, a default value set by the commondata::approach_offset_default is used.
in	*predict_window_food*	predict_window_food the size of the prediction window, i.e. how many steps back in memory are used to calculate the predicted food gain. This parameter is limited by the maximum commondata::history_size_perception value of the perception memory history size.
in	*time_step_model*	time_step_model optional time step of the model, overrides the value calculated from the spatial data.

8.3.3.19.1 Notable local variables

- consp_size - the size of the target conspecific,
- consp_mass - body mass of the target conspecific
- consp_dist - the distance to the target conspecific

target_position_agent - the target position of the agent, it does not coincide with the position of the target conspecific and is smaller by the value of the target offset.

- tmp_predator - temporary predator object, a subjective representation of the first nearest predator from the perception object of the actor agent.
- risk_pred_expect - an array keeping the expectancy of the predation risk for each predator in the perception object.
- n_pred_now - current number of predators in the perception object of the actor agent.
- body_mass_ratio - the ratio of the body mass of the actor agent to the target conspecific $\frac{M}{M_{TC}}$.
- food_gain_expect_baseline is a baseline expected food gain, not taking account of competition with the target conspecific.
- agent_length - agent length by condition::get_length() method.

8.3.3.19.1.1 Checks and preparations Check optional parameter for the food perception memory window. If the `predict_window_food` dummy parameter is not provided, its default value is the proportion of the whole perceptual memory window defined by commondata::history_perception_window_food. Thus, only the latest part of the memory is used for the prediction of the future food gain.
Check optional time step parameter. If unset, use global `commondata::global_time_step_model_current`.
Set the debug plot file name that will be passed to the predator-class-bound function the_environment::predator::risk_fish().

8.3.3.19.2 Implementation details

8.3.3.19.2.1 Get the properties of the target conspecific Get the properties of the conspecific from the perception object or real physical conspecific data. This is done by determining the `target_object` data type with "`select type`" construct (named construct `GET_TARGET`).
The distance to the target conspecific is determined from the target object with the_neurobio::conspec_percept_comp::get_dist() for perception object or the_environment::spatial::distance() for real conspecific.

- if the `target_object` is a conspecific from the perception object, its body length and mass are obtained from the respective data components of the_neurobio::conspec_percept_comp.
- if the `target_object` is real conspecific (the_neurobio::appraisal class), its body length and mass are obtained from lower order class component the_body::condition::get_length() and the_body::condition::get_mass() methods.

- in the case construct "default" case, if the `target_object` is neither a perception object nor real conspecific, get the location from the commondata::spatial class position data and other properties of the conspecific from the actor agent itself. Such a situation of **undefined target** type is unexpected and is likely to point to a bug. Therefore, an error is issued into the logger.

8.3.3.19.2.2 Determine the target offset Target offset `target_offset` can be provided as an optional dummy parameter to this procedure. However, if it is not provided explicitly, a default value is set as an average of the actor agent body length and the target conspecific body length.

8.3.3.19.2.3 Proximity check and target distance The agent approaches the conspecific but to a nonzero distance equal to the target offset value (`target_offset`). A check is done if the distance between the agent and the conspecific target object is actually smaller than the target offset.

- If so, the agent is already in close proximity to the target and there is no need to do an approach movement.

The approach distance is set to zero.

- The target position of the agent (`target_position_agent`) after such a zero approach actually coincides with the current position of the agent: it does not plan to swim.
- The expected cost of approach movement is also zero.
- If the agent is currently at a distance exceeding the target offset, the approach distance towards the target position of the actor agent is calculated as the true distance towards the target conspecific minus the offset value `target_offset`. (Note that whenever the default target offset is set, i.e. an average of the agent and target body sizes, the approach distance depends on the body sizes of both parties; it is also symmetric, i.e. the same if a large agent approaches a small target conspecific or *vice versa*.)
- Check if the distance to the target object exceeds the migration travel maximum value, set as commondata::migrate_dist_max_step body sizes of the agent. This case should not normally occur if the maximum distance is sufficiently large so that the target object is beyond the agent's visual range. So, nothing is done here except logging a warning.
- Calculate the prospective target position of the agent in proximity of the target conspecific `target_position_agent` with the offset, using the the_environment::offset_dist() procedure.
- Calculate expected cost of the swimming. The expected cost of swimming in the approach walk step depends on the above approach distance and is calculated using the the_body::condition::cost_swim() method assuming *laminar* flow (laminar flow is due to normal relatively slow swimming pattern).

8.3.3.19.2.4 Calculate expected risk of predation The expected risk of predation is assumed to **reduce** due to predator dilution or confusion effects if the agent approaches a conspecific. Furthermore, the risk values depend on the relative positions and distances between the predator and the actor agent and predator and the target conspecific.
Calculation of the expected risks of predation depends on the current perception of the agent. The simplest case is when the agent has currently **no predators** in its predator perception object:

- If there are no predators in the perception object, the expected general risk is calculated using the the_neurobio::predation_risk_backend() method assuming the current perception of predators is null.
- The expected direct risk of predation is zero if there are no predators in the current perception.

If there is a **non-zero number of predators** in the current predator perception, calculations of the expected risks are more complex.

General risk First, get the number of predators in the current perception object using the the_neurobio::percept_predator::get_count(
Accordingly, the **general risk** of predation taking account both the number of predators in the perception object and the average number of predators in the memory stack is calculated using the the_neurobio::predation_risk_backend() method. However, the expected number of predators is reduced by a factor defined by the parameter commondata::approach_conspecfic_dilute_general_risk (the integer expected number of predators is actually obtained by the `floor` intrinsic giving the lower integer value). (Therefore, the reduced expectancy is based on reduction of the expected number of predators while keeping memory part of the expectation fixed).

Direct risk Expectation of the direct risk of predation depends on the target position of the actor agent P_T (with the target offset Δ) and relative distances between the actor agent, target conspecific P_{TC} and all the predators P_i in the current perception object of the actor agent following the predicted agent movement.

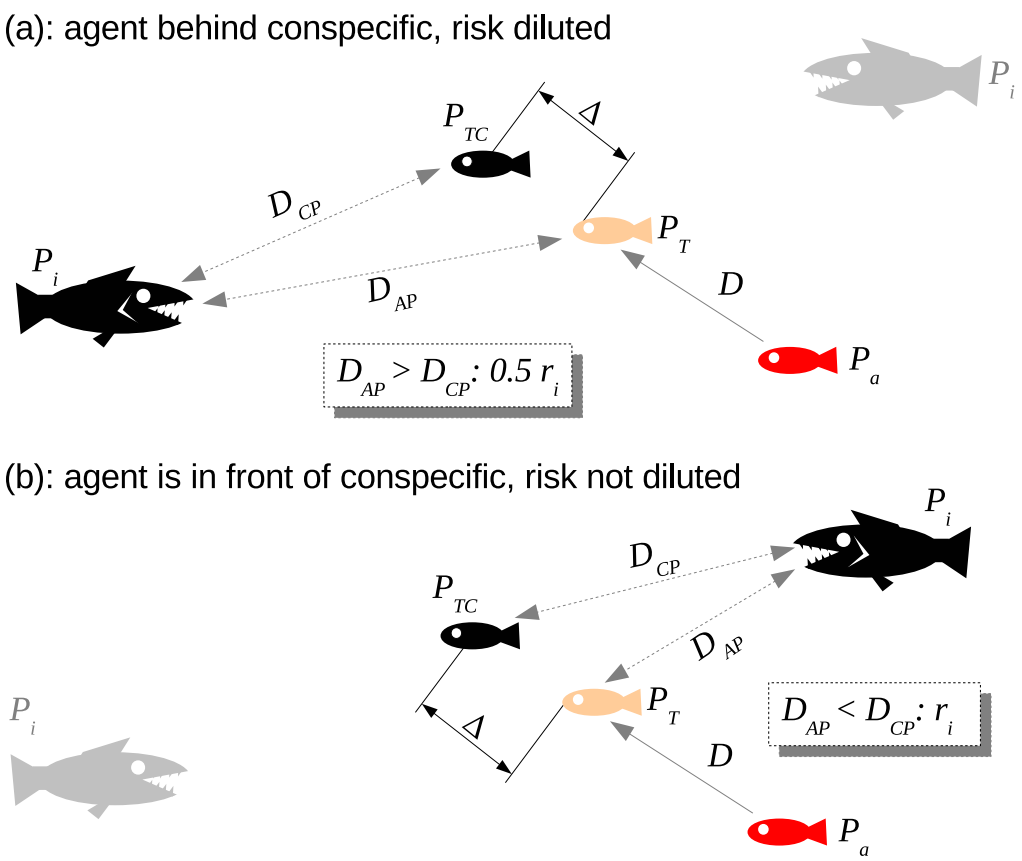


Figure 8.1 Calculation of the predicted direct risk of predation

First, allocate the array `risk_pred_expect` that keeps the values of risk for each of the predators in the perception object.
Then, cycle over all the predators P_i in the current perception object of the actor agent P_a and check if the prospective movement towards the target conspecific P_{TC} would place the agent *further* from the predator (a) than the target conspecific: $D_{AP} > D_{CP}$. If yes, direct risk of predation for this predator is equal to the risk of predation r unadjusted for the dilution or confusion effects multiplied by the commondata::approach_conspecfic_adjust_pair_↩ behind factor (normally 1/2 as diluted in a half by the target conspecific, $0.5r_i$). If the movement is likely to place the actor agent *closer* to the predator than the target conspecific $D_{AP} < D_{CP}$, the expected risk for the actor agent is calculated as unadjusted value r_i.
Thus, the predator dilution effect is introduced only if the actor agent is moving to the backward position further away from the predator (a) than the target conspecific (the target conspecific then is closer to the predator and suffers higher risk). If the actor agent moves to the forward position with respect to the predator (b), it suffers full unadjusted risk instead. This is the classical "selfish herd" effect.
Finally, the **maximum** value of the predation risks across all the predators $max(r_i)$ in the perception object of the actor agent constitutes the "final" expectation of the direct risk of predation: the_behaviour::approach_conspec::expected_pred_dir_ris

- At each (*i*-th) step of the loop, create a temporary the_environment::predator type object `tmp_predator` using the_environment::predator::make(). This predator's body size and the spatial position are obtained directly from the *i*-th predator 1/2 the agent's current perception object. But note that the agent is unable to determine the individually specific attack rate of the predator and uses the default value.

- If the distance between the agent and the *i*-th predator in the perception object (the temporary predator object `tmp_predator`) would become **shorter** than the distance between the target conspecific and the predator (i.e. the agent would go closer to the *i*-th predator than the target conspecific $D_{AP} < D_{CP}$), the direct risk of predation is calculated as unadjusted risk of predation computed using the

the_environment::predator::risk_fish() method, assuming the actor agent is in the target approach position `target_position_agent`.

- Otherwise, if the agent is going to relocate to a more remote location from the *i*-th predator ($D_{AP} > D_{CP}$), the baseline predation risk the_environment::predator::risk_fish() is diluted by a factor constant that is defined by the parameter commondata::approach_conspecfic_dilute_adjust_pair_behind (normally 1/2, i.e. diluted halfway by the target conspecific that is going to be closer to this predator).
- Finally, the value of the overall direct predation risk expected if the agent approaches the target conspecific is calculated as the maximum value of the expected risks across all predators in the perception object.
- The array of the expected direct risks from each of the predators in perception is logged out in the DEBUG mode.

8.3.3.19.2.5 Calculate the expected food gain The expected food gain is assumed to be **reduced** due to possible competition if the agent approaches a conspecific. Furthermore, the competition effect should depend on the relative body masses of the actor agent and the target conspecific.
First, a baseline assessment of the food gain f_0 is calculated that does not take into account any effects of competition with the target conspecific. It is equal to the average mass of all food items in the current food perception object weighted by the subjective probability of food item capture that is calculated based on the memory the_neurobio::perception::food_probability_capture_subjective(). (The mass is zero if there are no food items perceived).
The expected value of the food gain when the agent is about to approach the target conspecific is calculated as the baseline expected food gain f_0 multiplied by a nonparametric weighting function that depends on the ratio of the body mass of the actor agent M and the target conspecific M_{TC}:

$$f = f_0 \Phi(\frac{M}{M_{TC}}).$$

The function Φ is defined by the grid set by the arrays commondata::approach_food_gain_compet_factor_abscissa and commondata::approach_food_gain_compet_factor_ordinate.

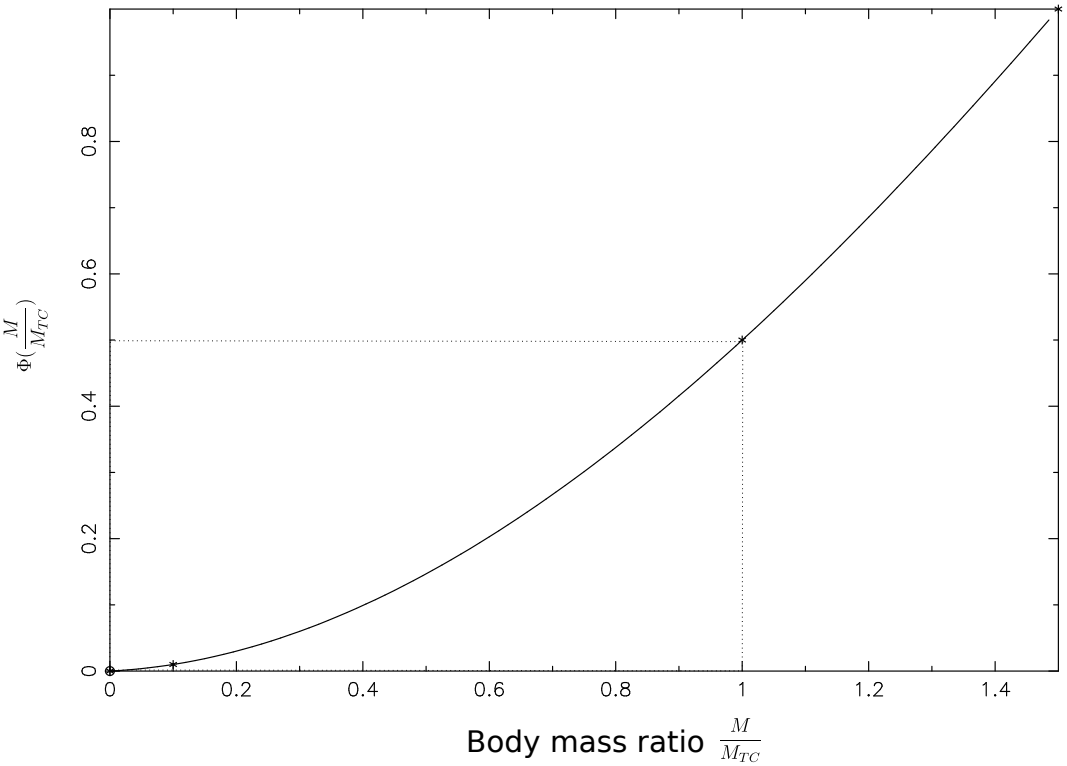


Figure 8.2 Food competition factor for expected food gain

Note

The maximum value of the grid abscissa defines the body mass ratio that guarantees 100% expectancy of winning of competition for food against the target conspecific. For example, the value of 1.5 means that an agent is guaranteed to get the whole baseline expected food gain if its body weight is 1.5 of the target conspecific. The grid ordinate corresponding to the abscissa 1.0 determines the food gain weighting when the body sizes of the agent and the target conspecifics are equal, e.g. 0.5 points to equal share by equal competitive ability.

Interpolation plots can be saved in the debug mode using this plotting command: `commondata::debug_interpolate_plot_`

Warning

Involves **huge** number of plots, should normally be disabled.

Definition at line 2622 of file m_behav.f90.

8.3.3.20 approach_conspecifics_motivations_expect()

```
subroutine the_behaviour::approach_conspecifics_motivations_expect (
             class(approach_conspec), intent(inout) this,
             class(appraisal), intent(in) this_agent,
             class(spatial), intent(in), optional target_object,
             real(srp), intent(in), optional target_offset,
             integer, intent(in), optional time_step_model,
             real(srp), intent(in), optional rescale_max_motivation )
```

`the_behaviour::approach_conspec::expectancies_calculate()` (re)calculates motivations from fake expected perceptions following from the procedure the_behaviour::approach_conspec::do_this().

Parameters

in	*this_agent*	this_agent is the actor agent which approaches a target conspecific.
in	*target_object*	target_object is the spatial target object the actor agent is going to approach.
in	*target_offset*	target_offset is an optional offset for the target, so that the target position of the approaching agent does not coincide with the target object. If absent, a default value set by the commondata::approach_offset_default is used.
in	*time_step_model*	time_step_model optional time step of the model, overrides the value calculated from the spatial data.
in	*rescale_max_motivation*	rescale_max_motivation optional maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

The probability of capture of the expected food object.
Expected food gain that is fitting into the stomach of the agent.

8.3.3.20.1 Notable local variables A full list of all perception overrides is available in the description of the the_neurobio::percept_components_motiv::motivation_components() procedure.

8.3.3.20.1.1 Perception overrides

- **perception_override_pred_dir** is the expected direct predation risk.

perception_override_predator is the expected general predation risk, that is based on a weighting of the current predation and predation risk from the memory stack.

- **perception_override_food_dir** is the expected number of food items in perception general predation.
- **perception_override_stomach** is the expected stomach contents as a consequence of approach movement. Note that there is no food consumption during approach.

- **perception_override_bodymass** is the expected body mass as a consequence of the approaching the target conspecific.
- **perception_override_energy** is the expected energy reserves as a consequence of the escape movement. Calculated from the body mass and weight.

8.3.3.20.2 Implementation details

8.3.3.20.2.1 Checks and preparations Check optional time step parameter. If not provided, use global parameter value from commondata::global_time_step_model_current.
Determine the target offset. Target offset `target_offset` can be provided as an optional dummy parameter to this procedure. However, if it is not provided explicitly, a default value is set as an average of the actor agent body length and the target conspecific body length. The the_neurobio::get_prop_size() method for polymorphic object gets the size of the target conspecific.

8.3.3.20.2.2 Call do_this As the first step, we use the **do**-procedure approach_conspec::do_this() to perform the behaviour desired without changing either the agent or its environment, obtaining the **subjective** values of the `this` behaviour components that later feed into the motivation **expectancy** functions:

- `perception_override_food_dir`
- `perception_override_pred_dir`
- `perception_override_predator`
- `perception_override_stomach`
- `perception_override_bodymass`
- `perception_override_energy`

8.3.3.20.2.3 Calculate expected (fake) perceptions

Fake perception of stomach content First, create a fake food item with the spatial position identical to that of the agent. The position is used only to calculate the illumination and therefore visual range. The cost(s) are calculated providing explicit separate distance parameter, so the zero distance from the agent is inconsequential. The size of the food item is obtained from the expected food gain by the reverse calculation function the_environment::mass2size_food(). Standard `make` method for the food item class is used.
Second, calculate the **probability of capture** of this expected food item. The probability of capture of the fake food item is calculated using the the_environment::food_item::capture_probability() backend assuming the distance to the food item is equal to the average distance of all food items in the **current perception** object. However, if the agent does not see any food items currently, the distance to the fake food item is assumed to be equal to the visibility range weighted by the (fractional) commondata::walk_random_dist_expect_food_uncertain_fact parameter. Thus, the expected *raw* food gain (in the `do`-function) is based on the past memory whereas the probability of capture is based on the latest perception experience.
Third, the expected food gain corrected for fitting into the agent's current stomach (and subtracting capture cost) is obtained by the_body::condition::food_fitting(). It is then weighted by the expected capture probability. Note that the probability of capture (weighting factor) is calculated based on the current perception (see above), but the travel cost is based on the actual expected %distance (see the_behaviour::walk_random::expectancies_calculate() for a similar procedure).
Stomach content: the perception override value for the stomach content is obtained incrementing the current stomach contents by the nonzero expected food gain, adjusting also for the digestion decrement (the_body::stomach_emptify_backend()).
Body mass: the **body mass** perception override is obtained by incrementing (or decrementing if the expected food gain is negative) the current body mass by the expected food gain and also subtracting the cost of living component.
Energy: The fake perception values for the energy reserves (`energy_override_perc`) using the `the_body::energy_reserve()` procedure.
Direct food perception: override is based on the current count of the food items in the perception object.

Note

Thus, the prediction of the food gain and stomach contents (see above) are based on a lower value that results from competition with the target conspecific. However, predicted perception of the general food availability is based on the current unmodified "objective" value.

Fake perception of predation risk **Predation risk**: finally, fake perceptions of predation risk are obtained from the values calculated in the `do` procedure: the_behaviour::approach_conspec::expected_pred_dir_risk and the_behaviour::approach_conspec::expected_predation_risk.

8.3.3.20.2.4 Calculate motivation expectancies The next step is to calculate the motivational expectancies using the fake perceptions to override the default (actual agent's) values. At this stage, first, calculate motivation values resulting from the behaviour done (the_behaviour::approach_conspec::do_this()) at the previous steps:

- what would be the motivation values *if* the agent does perform the_behaviour::approach_conspec?

Technically, this is done by calling the **neuronal response function**, `percept_components_motiv::motivation_compon` method, for each of the motivational states with `perception_override_` dummy parameters overriding the default values. Here is the list of the fake overriding perceptions for the `the_behaviour::approach_conspec` behaviour:

- `perception_override_food_dir`
- `perception_override_pred_dir`
- `perception_override_predator`
- `perception_override_stomach`
- `perception_override_bodymass`
- `perception_override_energy`

Real agent perception components are now substituted by the *fake* values resulting from executing this behaviour (`approach::do_this()` method). This is repeated for all the motivations: *hunger*, *passive avoidance, fear state* etc. These optional **override parameters** are substituted by the "fake" values.

8.3.3.20.2.5 Calculate primary and final motivations Next, from the perceptual components calculated at the previous step we can obtain the **primary** and **final motivation** values by weighed summing.
Here we can use global maximum motivation across all behaviours and perceptual components if it is provided, for rescaling.
Or can rescale using local maximum value for this behaviour only.
Transfer attention weights from the actor agent `this_agent` to the `this` behaviour component. So, we will now use the updated modulated attention weights of the agent rather than their default parameter values.
So the primary motivation values are calculated.
Primary motivations are logged in the debug mode.
There is **no modulation** at this stage, so the final motivation values are the same as primary motivations.

8.3.3.20.2.6 Calculate motivation expectancies Finally, calculate the finally **expected arousal level for this behaviour**. As in the GOS, the overall arousal is the maximum value among all motivation components.
Log also the final expectancy value in the debug mode.
Now as we know the expected arousal, we can choose the behaviour which would minimise this arousal level.
Definition at line 3109 of file m_behav.f90.

8.3.3.21 migrate_init_zero()

```
elemental subroutine the_behaviour::migrate_init_zero (
             class(migrate), intent(inout) this )
```

Initialise the **migrate** behaviour component to a zero state.

First init components from the base root class the_behaviour::behaviour_base. Mandatory label component that should be read-only.
The execution status is always FALSE, can be reset to TRUE only when the behaviour unit is called to execution.
And the *expectancy* components.
Abstract MOVE component.
Then init components of this specific behaviour component extended class.
Definition at line 3532 of file m_behav.f90.

8.3.3.22 migrate_do_this()

```
subroutine the_behaviour::migrate_do_this (
            class(migrate), intent(inout) this,
            class(appraisal), intent(in) this_agent,
            class(environment), intent(in) target_env,
            integer, intent(in), optional predict_window_food,
            integer, intent(in), optional predict_window_consp,
            integer, intent(in), optional predict_window_pred,
            integer, intent(in), optional time_step_model )
```

The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here MIGRATE).

Parameters

in	*this_agent*	this_agent is the actor agent which eats the food item.
in	*target_env*	target_env the target environment the actor agent is going to (e)migrate into.
in	*predict_window_food*	predict_window_food optional size of the *food* prediction window, i.e. how many steps back in memory are used to calculate the predicted food gain. This parameter is limited by the maximum commondata::history_size_perception value of the perception memory history size.
in	*predict_window_consp*	predict_window_consp optional size of the *conspecifics* prediction window, i.e. how many steps back in memory are used to calculate the predicted food gain. This parameter is limited by the maximum commondata::history_size_perception value of the perception memory history size.
in	*predict_window_pred*	predict_window_pred optional size of the *predator* prediction window, i.e. how many steps back in memory are used to calculate the predicted food gain. This parameter is limited by the maximum commondata::history_size_perception value of the perception memory history size.
in	*time_step_model*	time_step_model optional time step of the model, overrides the value calculated from the spatial data.

8.3.3.22.1 Notable variables

- **point_target_env** is the target point inside the target environment to which this agent is going to relocate.

distance_target is the distance to the target environment

- **mean_n_food_memory_old, mean_n_food_memory_new** are the average numbers of food items in the past memory window, the "older" and "newer" parts that are used to calculate the "older" $\overline{f_1}$ and "newer" $\overline{f_2}$ values of food availability retrieved from the perception memory. Used in calculation of the the_behaviour::hope function.
- **mean_size_food_memory_old, mean_size_food_memory_new** are the average sizes of food items in the past memory window, the "older" and "newer" parts that are used to calculate the "older" $\overline{f_1}$ and

"newer" $\overline{f_2}$ values of food availability retrieved from the perception memory. Used in calculation of the the_behaviour::hope function.

- **food_gain_memory_old, food_gain_memory_new** are the "older" $\overline{f_1}$ and "newer" $\overline{f_2}$ values of food availability retrieved from the perception memory. Used in calculation of the the_behaviour::hope function.
- **food_gain_memory_baseline** is the baseline value of the food gain retrieved from the memory, that is used to calculate the actual food gain expectancy value calculated from the hope function.
- **mean_n_pred_memory_old, mean_n_pred_memory_new** are the average numbers of predators in the past perception memory window.
- **pred_current** is the current estimate of the general predation risk.

8.3.3.22.2 Implementation details

8.3.3.22.2.1 Checks and preparations Check optional parameter for the food perception memory window. If the `predict_window_food` dummy parameter is not provided, its default value is its default value is the whole memory stack commondata::history_size_perception.
Check optional parameter for the conspecifics perception memory window. If the `predict_window_consp` dummy parameter is not provided, its default value is the whole memory stack commondata::history_size_perception.
Check optional parameter for the general predation risk perception memory window. If the `predict_window_pred` dummy parameter is not provided, its default value is the whole memory stack commondata::history_size_perception.
Check optional time step parameter. If unset, use global `commondata::global_time_step_model_current`.

8.3.3.22.2.2 Calculate the distance towards the target environment The distance towards the target environment (and the target point in this environment) is defined as the minimum distance towards all segments limiting this environment in the 2D X x Y projection

Warning

This is valid only for the simple box environment implementation. Generally, it equals to the minimum distance across all the polyhedrons limiting the target environment).

The target point for the migrating agent within the target environment is then not just the edge of the target environment, but some point penetrating inside to some distance defined by the parameter commondata::migrate_dist_penetrate_offset (in units of the agent's body length). The the_environment::environment::nearest_target() method is used to find the closest point in the target environment and the (smallest) distance towards this environment, these values are adjusted automatically for the offset parameter in the procedure call.
The distance value returned from the the_environment::environment::nearest_target() is saved into the this%distance data component and the target point (of class the_environment::spatial) is saved into the this%target_point data component.
Check if the distance to the target environment exceeds the migration travel maximum value, set as commondata::migrate_dist_max_step body sizes of the agent.

- So far nothing is done in such a case except logging a warning. Note that in the_behaviour::migrate::migrate_do_execute() method, agents that had the distance exceeding this threshold do a random correlated walk towards the target environment, but do not enter it.

8.3.3.22.2.3 Calculate expected cost of the swimming The expected cost of swimming in the random walk depends on the walk distance and is calculated using the the_body::condition::cost_swim() assuming *laminar* flow (laminar flow is due to normal relatively slow swimming pattern).

8.3.3.22.2.4 Calculate expected food gain The expected food gain resulting from emigrating into a completely different novel habitat cannot be assessed based only on current perception because the agent has virtually no information (i.e. no perception) about this habitat yet. The target habitat is a novel environment about which the agent has absolutely no local knowledge. A mechanism based on the **hope function** (the_behaviour::hope()) is used here. Specifically, the hope function calculates the expected food gain in the target novel habitat based on the ratio of the "newer" to "older" food gains in the perceptual memory of the agent.

Calculation of the "older" and "newer" average food gain values from the memory involves several steps. First, average *number* of food items and the average *size* of the food items in the above two halves of the memory stack is calculated using the the_neurobio::memory_perceptual::get_food_mean_n_split() and the_neurobio::memory_perceptual::get_food_mean_size_split() procedures. (Note that the `split_val` parameter to this procedure is not provided so the default 1/2 split is used.)
Second, the values of the "old" and "new" *food gain* used to calculate the expectations are obtained by weighting the respective average mass of the food item by the average number of food items if this number is less than 1 or 1 (i.e. unweighted) if their average number is higher.

$$\begin{cases} f_1 = \overline{m_1} \cdot \overline{n_1}, & \overline{n_1} < 1 \\ f_1 = \overline{m_1}, & \overline{n_1} \geq 1 \end{cases} \qquad \begin{cases} f_2 = \overline{m_2} \cdot \overline{n_2}, & \overline{n_2} < 1 \\ f_2 = \overline{m_2}, & \overline{n_2} \geq 1 \end{cases}$$

where $\overline{m_1}$ is the average mass of the food items and $\overline{n_1}$ is the average number of food items in the "older" half of the perceptual memory stack and $\overline{m_2}$ is the average mass of the food items and $\overline{n_2}$ is the average number of food items in the "newer" half of the memory stack.
Thus, if the agent had some relatively poor perceptual history of encountering food items, so that the average *number* of food items is fractional < 1 (e.g. average number 0.5, meaning that it has seen a single food item approximately every other time step), the food gain is weighted by this fraction (0.5). If, on the other hand, the agent had more than one food items at each time step previously, the average food item size is unweighted (weight=1.0). This conditional weighting reflects the fact that it is not possible to eat more than one food item at a time in this model version.

Note

A similar expectancy assessment mechanism is used in the assessment of the food gain expectancy for the the_behaviour::walk_random behaviour component the_behaviour::walk_random_do_this().

The next step is to calculate the baseline food gain f_0, against which the expectancy based on the the_behaviour::hope() function is evaluated. This baseline value is obtained by weighting the average mass of the food items in the whole memory stack $\overline{m}$ by their average number $\overline{n}$ provided this number is *n<1* as above:

$$\begin{cases} f_0 = \overline{m} \cdot \overline{n}, & \overline{n} < 1; \\ f_0 = \overline{m}, & \overline{n} \geq 1 \end{cases}$$

This baseline value is then weighted by the subjective probability of food item capture that is calculated based on the memory the_neurobio::perception::food_probability_capture_subjective().
Finally, the the_behaviour::hope() function is called with the above estimates for the baseline food gain, its "older" and "newer" values. The *zero hope ratio* and the *maximum hope* parameters are obtained from commondata::migrate_food_gain_ratio_zero_hope and commondata::migrate_food_gain_maximum_hope parameter constants.

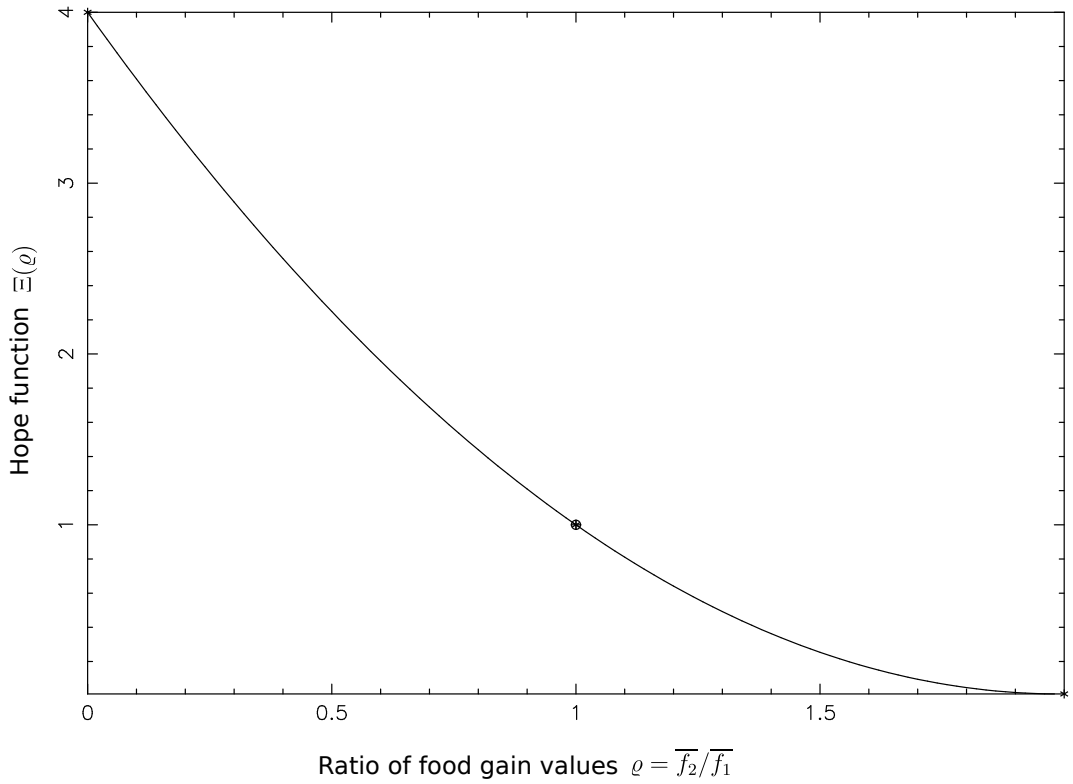


Figure 8.3 The hope function

8.3.3.22.2.5 Calculate expected food items perception A similar, although simpler, procedure based on the the_behaviour::hope function as above is used to calculate the expected *number* of food items perceived in the target novel habitat.
Here, the baseline value f_0 is the current number of food items in the food perception object, and the historical ratio ϱ is calculated as the mean number of food items in the old to new memory parts:

$$\varrho = \frac{\overline{n_2}}{\overline{n_1}}.$$

The *zero hope ratio* and the *maximum hope* parameters are also obtained from commondata::migrate_food_gain_ratio_zero_hope and commondata::migrate_food_gain_maximum_hope parameter constants.

8.3.3.22.2.6 Calculate expected predation risks **Direct predation** risk is assumed to be zero for migration.
General predation risk expectancy is not possible to determine because there is no local perception of the target novel environment yet. Therefore, its assessment is based on the the_behaviour::hope() function, just as the expected food gain.

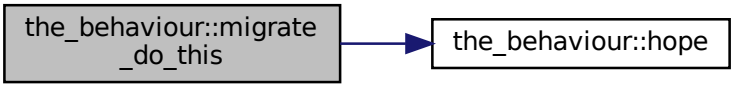
- First, calculate the older and newer predation averages from the memory stack;

Second, calculate the *current* general risk of predation, based on the local perception. This is done calling the the_neurobio::predation_risk_backend() function. This current risk serves as a baseline value (f_0) for calculation of the general risk in the target novel environment.

- Third, the expectancy value of general predation risk in the target novel environment is obtained via the the_behaviour::hope() function. If the general predation risk is increasing in the local environment, its expectancy in the novel environment diminishes, if the risk is reducing over time in the local environment, the novel environment expectancy increases. The hope grid values for the general predation hope function are defined by the commondata::migrate_predator_zero_hope and commondata::migrate_predator_maximum_hope parameter constants.

8.3.3.22.2.7 Calculate expected conspecifics The expected number of conspecifics in the target environment is calculated as an average retrieved from the memory stack with the memory window defined by `predict_↩ window_consp_here`.

Definition at line 3567 of file m_behav.f90.
Here is the call graph for this function:



8.3.3.23 migrate_motivations_expect()

```
subroutine the_behaviour::migrate_motivations_expect (
            class(migrate), intent(inout) this,
            class(appraisal), intent(in) this_agent,
            class(environment), intent(in) target_env,
            integer, intent(in), optional predict_window_food,
            integer, intent(in), optional predict_window_consp,
            integer, intent(in), optional predict_window_pred,
            integer, intent(in), optional time_step_model,
            real(srp), intent(in), optional rescale_max_motivation )
```

the_behaviour::migrate::expectancies_calculate() (re)calculates motivations from fake expected perceptions following from the procedure migrate::do_this().

Parameters

in	*this_agent*	this_agent is the actor agent which is going to migrate.
in	*target_env*	target_env the target environment the actor agent is going to (e)migrate into.
in	*predict_window_food*	predict_window_food optional size of the *food* prediction window, i.e. how many steps back in memory are used to calculate the predicted food gain. This parameter is limited by the maximum commondata::history_size_perception value of the perception memory history size.
in	*predict_window_consp*	predict_window_consp optional size of the *conspecifics* prediction window, i.e. how many steps back in memory are used to calculate the predicted food gain. This parameter is limited by the maximum commondata::history_size_perception value of the perception memory history size.
in	*predict_window_pred*	predict_window_pred optional size of the *predator* prediction window, i.e. how many steps back in memory are used to calculate the predicted food gain. This parameter is limited by the maximum commondata::history_size_perception value of the perception memory history size.
in	*time_step_model*	time_step_model optional time step of the model, overrides the value calculated from the spatial data.
in	*rescale_max_motivation*	rescale_max_motivation optional maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

Expected food gain that is fitting into the stomach of the agent.
The probability of capture of the expected food object.

8.3.3.23.1 Notable local variables

8.3.3.23.1.1 Perception overrides

- **perception_override_conspec** is the expected number of conspecifics.

perception_override_pred_dir is the expected direct predation risk.

- **perception_override_predator** is the expected general predation risk, that is based on a weighting of the current predation and predation risk from the memory stack.
- **perception_override_food_dir** is the expected number of food items in perception.
- **perception_override_stomach** is the expected stomach contents as a consequence of random walk.
- **perception_override_bodymass** is the expected body mass as a consequence of the random walk.
- **perception_override_energy** is the expected energy reserves as a consequence of the escape movement. Calculated from the body mass and weight.

8.3.3.23.2 Implementation details

8.3.3.23.2.1 Checks and preparations Check optional parameter for the food perception memory window. If the `predict_window_food` dummy parameter is not provided, its default value is its default value is the whole memory stack commondata::history_size_perception.
Check optional parameter for the conspecifics perception memory window. If the `predict_window_consp` dummy parameter is not provided, its default value is the whole memory stack commondata::history_size_perception.
Check optional parameter for the general predation risk perception memory window. If the `predict_window_↩ pred` dummy parameter is not provided, its default value is the whole memory stack commondata::history_size_perception.
Check optional time step parameter. If unset, use global commondata::global_time_step_model_current.

8.3.3.23.2.2 Call do_this As the first step, we use the **do**-procedure `migrate::do_this()` => `the_behaviour::walk_random_do_this()` to perform the behaviour desired without changing either the agent or its environment, obtaining the **subjective** values of the `this` behaviour components that later feed into the motivation **expectancy** functions:

- `perception_override_food_dir`
- `perception_override_conspec`
- `perception_override_pred_dir`
- `perception_override_predator`
- `perception_override_stomach`
- `perception_override_bodymass`
- `perception_override_energy`

8.3.3.23.2.3 Calculate expected (fake) perceptions First, create a fake food item with the spatial position identical to that of the agent. The position is used to calculate the current illumination and therefore visual range. The cost(s) are calculated providing explicit separate distance parameter. The size of the food item is obtained from the expected food gain by the reverse calculation function the_environment::mass2size_food(). Standard `make` method for the food item class is used.
Second, calculate the **probability of capture** of this expected food item. The probability of capture of the fake food item is calculated using the the_environment::food_item::capture_probability() backend assuming the distance to the food item is equal to the average distance of all food items in the **current perception** object. However, if the agent does not see any food items currently, the distance to the fake food item is assumed to be equal to the visibility range weighted by the (fractional) commondata::dist_expect_food_uncertain_fact parameter.
Third, the expected food gain corrected for fitting into the agent's stomach and capture cost is obtained by the_body::condition::food_fitting(). It is then weighted by the expected capture probability.

Stomach content: the perception override value for the stomach content is obtained incrementing the current stomach contents by the nonzero expected food gain, adjusting also for the digestion decrement (the_body::stomach_emptify_backend()).
Body mass: the **body mass** perception override π_m is obtained by incrementing (or decrementing if the expected food gain is negative) the current body mass M by the expected food gain ϕ and also subtracting the cost of living M_c (the_body::condition::living_cost()) and the expected cost of movement into the target novel habitat μ:

$$\pi_m = M + \phi - M_c - \mu$$

Thus, probability of capture and costs of food processing in calculating the stomach content increment depend on the distance to the expected food item and do not take into account the travel cost to the novel environment (it can be quite large, beyond the visibility of the expected food item). However, expectancy of the body mass (the fake perception value) takes into account the cost of migration movement to the novel target habitat.
Energy: The fake perception values for the energy reserves (`energy_override_perc`) using the the_body::energy_reserve() procedure.
Direct food perception: The fake perception of the number of food items expected for the perception in the target novel environment is calculated from the this%expected_food_dir component (obtained in the `do_this` procedure).
Predation risk: fake perceptions of predation risk are obtained from the values calculated in the `do` procedure: the_behaviour::migrate::expected_pred_dir_risk and the_behaviour::migrate::expected_predation_risk.
Number of conspecifics: finally, the fake perception of the number of conspecifics is calculated from the values calculated in the `do` procedure: the_behaviour::migrate::expected_consp_number.

8.3.3.23.2.4 Calculate motivation expectancies The next step is to calculate the motivational expectancies using the fake perceptions to override the default (actual agent's) values. At this stage, first, calculate motivation values resulting from the behaviour done (`migrate::do_this()`) at the previous steps: what would be the motivation values *if* the agent does perform MIGRATE? Technically, this is done by calling the **neuronal response function**, `percept_components_motiv::motivation_components()` method, for each of the motivational states with `perception_override_` dummy parameters overriding the default values. Here is the list of the fake overriding perceptions for the MIGRATE behaviour:

- `perception_override_food_dir`
- `perception_override_conspec`
- `perception_override_pred_dir`
- `perception_override_predator`
- `perception_override_stomach`
- `perception_override_bodymass`
- `perception_override_energy`

Real agent perception components are now substituted by the *fake* values resulting from executing this behaviour (`reproduce::do_this()` => `the_behaviour::reproduce_do_this()` method). This is repeated for all the motivations: *hunger*, *passive avoidance, active avoidance* etc. These optional **override parameters** are substituted by the "fake" values.

8.3.3.23.2.5 Calculate primary and final motivations Next, from the perceptual components calculated at the previous step we can obtain the **primary** and **final motivation** values by weighed summing.
Here we can use global maximum motivation across all behaviours and perceptual components if it is provided, for rescaling.
Or can rescale using local maximum value for this behaviour only.
Transfer attention weights from the actor agent `this_agent` to the `this` behaviour component. So, we will now use the updated modulated attention weights of the agent rather than their default parameter values.
So the primary motivation values are calculated.
Primary motivations are logged in the debug mode.
There is **no modulation** at this stage, so the final motivation values are the same as primary motivations.

8.3.3.23.2.6 Calculate motivation expectancies Finally, calculate the finally **expected arousal level for this behaviour**. As in the GOS, the overall arousal is the maximum value among all motivation components.
Log also the final expectancy value in the debug mode.
Now as we know the expected arousal, we can choose the behaviour which would minimise this arousal level.
Definition at line 3992 of file m_behav.f90.

8.3.3.24 migrate_do_execute()

```
subroutine the_behaviour::migrate_do_execute (
             class(migrate), intent(inout) this,
             class(appraisal), intent(inout) this_agent,
             class(environment), intent(in) target_env )
```

Execute this behaviour component "migrate" by `this_agent` agent.

Parameters

`in,out`	*this_agent*	[in] this_agent is the actor agent which goes down.
`in`	*target_env*	target_env the target environment the actor agent is going to (e)migrate into.

8.3.3.24.1 Implementation details

8.3.3.24.1.1 Step 1: do_this First, we use the intent-in **do**-procedure the_behaviour::migrate::do_this() to perform the behaviour desired. However, instead of expectations, get the target point in the novel habitat.(Expectancies for food gain, predator risk etc. are not used at this stage, memory windows are absent from the parameter list.)

8.3.3.24.1.2 Step 2: Change the agent

Try to relocate to the target novel habitat The agent does a directional walk at this%distance towards the this%target_point in the novel target environment. However, it is possible only if the walk distance does not exceed the maximum value defined by the commondata::migrate_dist_max_step body sizes of the agent.

- If this is the case, a warning is logged.
- the agent is executing a Gaussian correlated random walk towards the target point. The average walk length is the above maximum distance minus 95% confidence limit and the CV is the default for random walks (thus, there is almost a guarantee that the actual walk is the maximum commondata::migrate_dist_max_step distance and unlikely to exceed it. This walk is, additionally, limited to the present environment (i.e. no migration is performed by the agent).

If the above limit on the length of a single walk is not exceeded, the agent relocates to the target point in the novel target environment. It is now in the target environment.
In the DEBUG Mode, print diagnostic information to the logger.

Process the cost of movement

- Reset the body mass of the actor agent subtracting the actual cost of the migration moving that is automatically calculated in the call to the_body::condition::cost_swim(). The the_body::condition::set_mass() method is used here to adjust the mass.

Additionally, also call the `the_body::condition::set_length()` method to update the body length history stack. However, the value_set parameter here is just the current value. This fake re-setting of the body length is done to keep both mass and length synchronised in their history stack arrays (there is no procedure for only updating history).

- After resetting the body mass, update energy reserves of the agent, that depend on both the length and the mass.

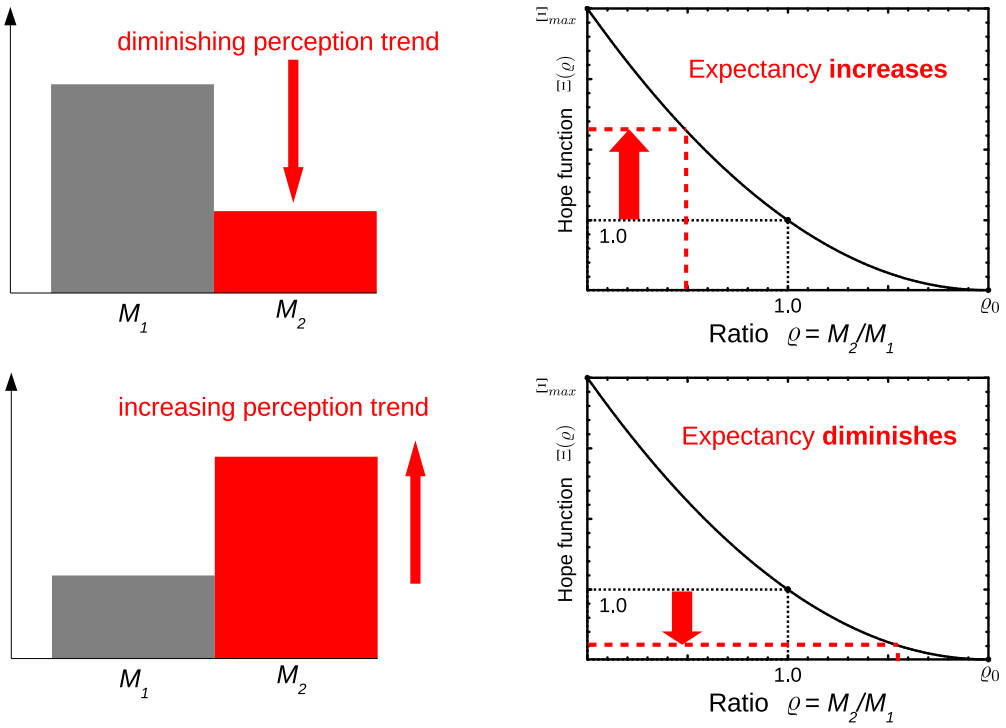
Finally, check if the agent is starved to death. If yes, the agent can die without going any further.

8.3.3.24.1.3 Step 3: Change the environment Random walk does not affect the environmental objects.
Definition at line 4467 of file m_behav.f90.

8.3.3.25 hope()

```
pure real(srp) function the_behaviour::hope (
            real(srp), intent(in) baseline,
            real(srp), intent(in) memory_old,
            real(srp), intent(in) memory_new,
            real(srp), intent(in), optional zero_hope,
            real(srp), intent(in), optional maximum_hope,
            real(srp), dimension(:), intent(in), optional raw_grid_x,
            real(srp), dimension(:), intent(in), optional raw_grid_y )
```

The hope function for the assessment of expectancy for a completely novel stimulus or environment for which local information is absent.
Calculation of the expectancy and therefore fake perceptions is not possible for completely novel environment or stimuli (e.g. for emigrating into a completely different novel habitat) based on the current perception because the agent has absolutely no local information (i.e. no perception of this habitat yet).
A mechanism based on the **hope function** should be used in such a case.



Expectancy value $F_{exp} = f_0 \cdot \Xi(\varrho)$
(where f_0 is the baseline expectancy based on available information)

Example: The agent accesses a trend of food perception from its memory using the ratio of "newer" to "older" values M_2/M_1.

- If the food availability is diminishing $M_2/M_1<1$, the expectancy of the food gain in the target novel habitat increases.
- If the food availability trend is increasing, the expectancy of the food gain in the target novel environment decreases.

Figure 8.4 The hope function mechanism

- A baseline expectancy f_0 based on the locally available information (e.g. local expectation of the food gain) is selected.
- Then, a trend of the baseline expectancy characteristic (e.g. average food gain) in the past memory stack is determined by

- splitting a food memory stack *window* into two halves: older $\mathbf{M_1}$ and newer $\mathbf{M_2}$,
- calculating the average local expectancies for the older $\overline{f_1}$ and newer $\overline{f_2}$ parts,
- calculating the ratio

$$\varrho = \frac{\overline{f_2}}{\overline{f_1}}.$$

Following this, the expectancy (e.g. expected food gain) for the novel stimuli or situation is calculated as:

$$F_{exp} = f_0 \cdot \Xi(\varrho),$$

where f_0 is the baseline food gain against which the expectancy is evaluated, and $\Xi(\varrho)$ is the "hope" function that is obtained as a nonparametric relationship (see the right panel plots above): nonlinear interpolation based on the grid vectors $\mathbf{V}$ and $\mathbf{W}$:

$$\mathbf{V} = \begin{pmatrix} 0.0 \\ 1.0 \\ \varrho_0 \end{pmatrix}, \mathbf{W} = \begin{pmatrix} \Xi_{max} \\ 1.0 \\ \rightarrow 0.0 \end{pmatrix}$$

where ϱ_0 is the *zero hope ratio* parameter and Ξ_{max} is the *maximum hope* parameter.

Parameters

in	*baseline*	baseline is the baseline stimulus expectancy f_0 that is based on the locally available information.
in	*memory_old*	memory_old is the older part (half) of the memory stack $\overline{f_1}$ for the baseline perception.
in	*memory_new*	memory_new is the newer part (half) of the memory stack $\overline{f_2}$ for the baseline perception.
in	*zero_hope*	zero_hope is the zero hope ratio ϱ_0 parameter of the hope function grid abscissa vector.
in	*maximum_hope*	maximum_hope is the maximum hope Ξ_{max} parameter of the hope function grid ordinate vector.
in	*raw_grid_x*	raw_grid_x a raw interpolation grid array that can be provided (along with raw_grid_y) instead of the normal `zero_hope` and `maximum_hope` parameters.
in	*raw_grid_y*	raw_grid_y a raw interpolation grid array that can be provided (along with raw_grid_x) instead of the normal `zero_hope` and `maximum_hope` parameters.

Returns

The expected value for the wholly novel stimulus or environment.

Note

Note that the scalar parameters `zero_hope` and `maximum_hope` represent the normal standard way to provide the interpolation grid for the hope function. However, these grids can also be accepted as raw grid arrays (see `raw_grid_x` and `raw_grid_y` parameters below).

Raw grid arrays have priority if both raw grid arrays and normal scalar parameters `zero_hope` and `maximum_hope` are simultaneously provided.

Warning

The grid vectors `raw_grid_x` and `raw_grid_y` must have the same length.

8.3.3.25.1 Notable variables

- **memory_ratio** is the ratio of the newer to older memory values;

hope_func_grid_abscissa and **hope_func_grid_ordinate** are the hope function grid arrays. They define the nonparametric hope function that is obtained by nonlinear interpolation. These arrays can be also provided as raw `raw_grid_x` `raw_grid_y` parameters.

Implementation details First, calculate the memory-based ratio

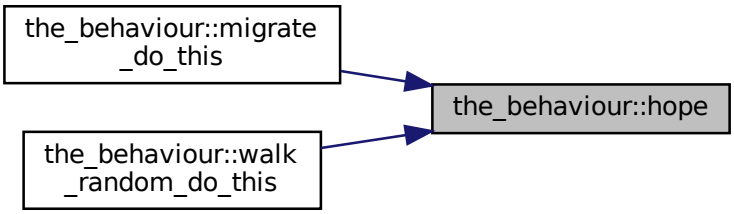
$$\varrho = \frac{\overline{f_2}}{\overline{f_1}}.$$

- The calculation also checks for possible division by zero, if the older memory value $\overline{f_2} = 0.0$; in such a case, the ratio is set to the maximum abscissa grid value resulting in zero hope function.

An additional case of both $\overline{f_1} = 0.0$ and $\overline{f_2} = 0.0$ is also checked, the ratio in such a case is set to 1.0, bringing about a unity hope function value (i.e. baseline expectancy is unchanged).
Second, get the hope function grid vectors $\mathbf{V}$ and $\mathbf{W}$ as:

```
V = [ 0.0_SRP,      1.00_SRP,  zero_hope ]
W = [ maximum_hope, 1.00_SRP,       ZERO ]
```

Finally, the hope function value is obtained from a nonlinear interpolation based on `DDPINTERPOL` (see HEDTOOLS) with the interpolation grid defined by the $\mathbf{V}$ (abscissa) and $\mathbf{W}$ (ordinate) vectors.
If neither a pair of the scalar parameters `zero_hope` and `maximum_hope` nor the raw grid arrays `raw_grid_x` and `raw_grid_y` are provided, return commondata::missing value as an indicator of error.
Definition at line 4656 of file m_behav.f90.
Here is the caller graph for this function:



8.3.3.26 depth_walk_default()

```
elemental real(srp) function the_behaviour::depth_walk_default (
            real(srp), intent(in) length,
            real(srp), intent(in), optional walk_factor )
```

Calculate the default upward and downward walk step size. This function is called from the_behaviour::go_down_do_this() and the_behaviour::go_down_motivations_expect() if the upwards or downwards walk size is not provided explicitly.

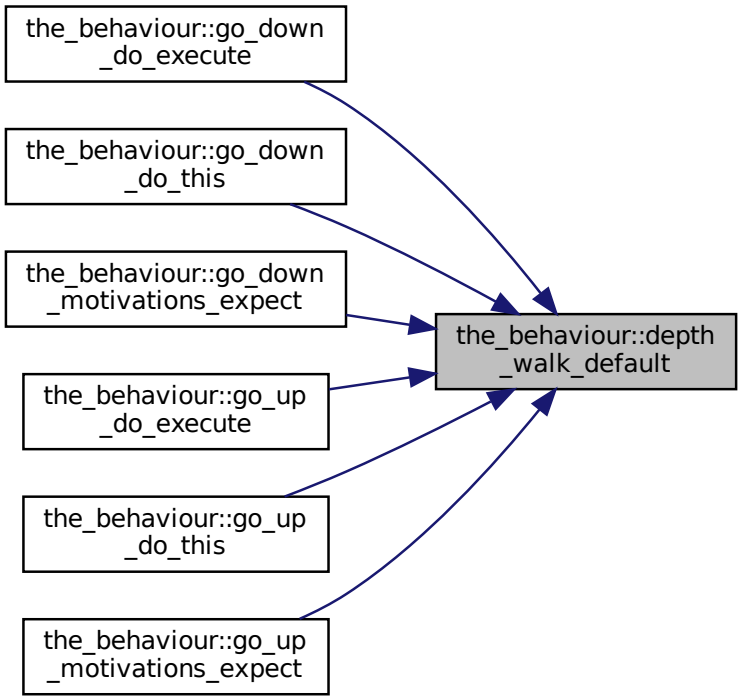
Parameters

in	*length*	length The body length of the agent.
in	*walk_factor*	walk_factor The multiplocation factor for the walk step. The fdefault value is defined by the parameter commondata::up_down_walk_step_stdlength_factor.

Returns

The default up/down walk step size.

8.3.3.26.1 Details If the walk size is not provided, it is set equal to the agent's body length multiplied by the commondata::up_down_walk_step_stdlength_factor factor parameter.
Definition at line 4781 of file m_behav.f90.
Here is the caller graph for this function:



8.3.3.27 go_down_depth_init_zero()

```
elemental subroutine the_behaviour::go_down_depth_init_zero (
            class(go_down_depth), intent(inout) this )
```

Initialise the **go down to a deeper spatial layer** behaviour component to a zero state.
First init components from the base root class the_behaviour::behaviour_base. Mandatory label component that should be read-only.
The execution status is always FALSE, can be reset to TRUE only when the behaviour unit is called to execution.
And the *expectancy* components.
Abstract MOVE component.
Then init components of this specific behaviour component extended class.
Definition at line 4807 of file m_behav.f90.

8.3.3.28 go_down_do_this()

```
subroutine the_behaviour::go_down_do_this (
            class(go_down_depth), intent(inout) this,
```

```
class(appraisal), intent(in) this_agent,
real(srp), intent(in) max_depth,
real(srp), intent(in), optional depth_walk,
integer, intent(in), optional predict_window_food,
integer, intent(in), optional time_step_model )
```

Do go down by `this_agent` (the actor agent). Subjective assessment of the motivational value for this is based on the number of food items, conspecifics and predators at the layers below the `this_agent` actor agent.

Parameters

`in,out`	*this*	[inout] this the object itself.
`in`	*this_agent*	this_agent is the actor agent which goes down.
`in`	*max_depth*	max_depth is the maximum limit on the depth.
`in`	*depth_walk*	depth_walk Optional downward walk size, by how deep the agent goes down.
`in`	*predict_window_food*	predict_window_food the size of the prediction window, i.e. how many steps back in memory are used to calculate the predicted food gain. This parameter is limited by the maximum commondata::history_size_perception value of the perception memory history size.
`in`	*time_step_model*	time_step_model optional time step of the model, overrides the value calculated from the spatial data.

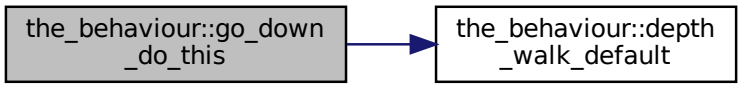
8.3.3.28.1 Implementation details First, check if the size of the downward walk `depth_walk` dummy parameter is provided.
If it is not provided, it is set equal to the agent's body length multiplied by the commondata::up_down_walk_step_stdlength_factor factor parameter. Calculated by `the_behaviour::depth_walk_default()`.
Check optional parameter for the food perception memory window. If the `predict_window_food` dummy parameter is not provided, its default value is the proportion of the whole perceptual memory window defined by commondata::history_perception_window_food. Thus, only the latest part of the memory is used for the prediction of the future food gain.
Check optional time step parameter. If unset, use global `commondata::global_time_step_model_current`.

8.3.3.28.1.1 Downward step size Here, first, check if the target depth is likely to go beyond the environment depth limits and reduce the downward walk step size accordingly. Namely, if the depth coordinate of the actor agent plus the depth step exceeds the maximum depth, the step is reduced to be within the available environment: $D_{max} - d_a - \varepsilon$, where D_{max} is the maximum depth, d_a is the agent's current depth and ε is a very small constant defined by the parameter commondata::zero.
The down step size component of the class is then equal to the `depth_walk`.

8.3.3.28.1.2 The cost of swimming down The expected cost of the swimming down by the buoyancy is much smaller than active propulsion. It is set as a fraction, defined by the parameter commondata::swimming_cost_factor_buoyancy_down, of active laminar propulsion calculated by function the_body::condition_cost_swimming_burst().

8.3.3.28.1.3 Calculate expected perceptions Calculate the number of conspecifics at the down of the agent using the function perception::consp_below().
Calculate the expected predation risk at the down of the agent using the `the_neurobio::predation_risk_backend()` function. This is a general predation risk (the_neurobio::percept_components_motiv::predator), not direct risk based on the distance to the nearest predator (see the_neurobio::percept_components_motiv::pred_dir).
Calculate the expected food gain as an average mass of the food items down the agent. It is used by calling perception::food_mass_below() function. This expected food gain is then weighted by the subjective probability of food item capture that is calculated based on the memory the_neurobio::perception::food_probability_capture_subjective().
Definition at line 4838 of file m_behav.f90.

Here is the call graph for this function:



8.3.3.29 go_down_motivations_expect()

```
subroutine the_behaviour::go_down_motivations_expect (
             class(go_down_depth), intent(inout) this,
             class(appraisal), intent(in) this_agent,
             real(srp), intent(in), optional depth_walk,
             real(srp), intent(in), optional max_depth,
             class(environment), dimension(:), intent(in), optional environments,
             integer, intent(in), optional time_step_model,
             real(srp), intent(in), optional rescale_max_motivation )
```

`go_down_depth::motivations_expect()` is a subroutine (re)calculating motivations from fake expected perceptions following from the procedure `go_down_depth::do_this()` => `the_behaviour::go_down_do_this()`.

Parameters

`in,out`	*this*	[inout] this the object itself.
`in`	*this_agent*	this_agent is the actor agent which goes down.
`in`	*depth_walk*	depth_walk The downward walk size, by how deep the agent goes down.
`in`	*max_depth*	max_depth is the optional maximum limit on the depth.
`in`	*environments*	environments optional array of the all available environments where the this agent can be in, needed for the calculation of the depth limits. If such an array of the environments is provided, `max_depth` has precedence.
`in`	*time_step_model*	[in] time_step_model optional time step of the model, **overrides** the value calculated from the spatial data.
`in`	*rescale_max_motivation*	rescale_max_motivation maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

Target depth, i.e. the absolute depth of the agent after it moves down.

8.3.3.29.1 Notable local variables

8.3.3.29.1.1 Perception overrides

- **expect_food_perc_override** is the fake perception for the food items at the target depth.

expect_depth_perc_override is the fake perception of the depth, identical to the target depth.

- **expect_light_perc_override** is the fake perception of the illumination level at the target depth.
- **expect_mass_perc_override** is the fake perception value for the mass from the expected food.

- **expect_stomach_perc_override** is the fake perception value for the stomach increment from the expected food.
- **expect_energy_perc_override** is the fake perception for the energy reserves from the expected food at the target depth.
- **expected_probability_capture** is the expected probability of capture of the expected food item at the target depth.
- **expect_conspecicifc_perc_override** is the fake perception value for the number of conspecifics at the target depth.
- **expect_predator_perc_override** is fake perception value for the predation risk at the target depth.

8.3.3.29.2 Implementation details

8.3.3.29.2.1 Sanity checks and preparations Initially, check if the size of the downward walk `depth_walk` dummy parameter is provided.
If it is not provided, it is set equal to the agent's body length multiplied by the commondata::up_down_walk_step_stdlength_factor factor parameter. Calculated by `the_behaviour::depth_walk_default()`.
Check downward step size. Here, first, check if the target depth is likely to go beyond the environment depth limits and reduce the downward walk step size accordingly. Either the explicitly provided maximum depth dummy parameter `max_depth` or an array of possible environment objects where the `this_agent` actor agent can be located is used to get the depth limit.
If the array of possible environment objects that can contain the actor agent is provided, the check involves the `the_environment::spatial::find_environment()` function to find the specific environment object the agent is currently in followed by `the_environment::environment::depth_max()` to find the minimum depth in this environment object.
If the array of possible environment objects that can contain the actor agent is not provided, the current environment is obtained from the global array the_environment::global_habitats_available. In this case, the environment that actor agent is within is determined using the the_environment::spatial::find_environment() method, which is in followed by the_environment::environment::depth_max()` to find the minimum depth in this environment object.
If `max_depth` is provided, it has precedence over the depth detected explicitly or implicitly from the environment objects.
In the case the maximum depth cannot be determined,it is set as the depth of the actor agent (with an additional condition that it should exceed zero), so movement down would be **impossible**.
If the depth coordinate of the actor agent plus the depth step exceeds the maximum depth, the step is reduced to be strictly within the available environment. However, it should also never be below zero.
Check optional time step parameter. If not provided, use global parameter value from commondata::global_time_step_model_current.
Assess the number of food items below using the perception::food_items_below() method.
Calculate the expected distance to the food item. It is equal to the average distance to the food items perceived below in case there are any such items perceived below. Calculated using the perception::food_dist_below() method. However, if there are no food items below (resulting a commondata::missing distance, see perception::food_dist_below()), the expected distance is set the downward walk distance `depth_walk`, that should be sufficiently long to assure the probability of food item capture is very small or zero.

8.3.3.29.2.2 Call do_this As the first step, we use the **do**-procedure `go_down_depth::do_this()` => `the_behaviour::go_down_do_this()` to perform the behaviour desired without changing either the agent or its environment, obtaining the **subjective** values of the `this` behaviour components that later feed into the motivation **expectancy** functions:

- `perception_override_light`
- `perception_override_depth`
- `perception_override_food_dir`
- `perception_override_predator`
- `perception_override_stomach`

- perception_override_bodymass
- perception_override_energy

The absolute value of the target depth is equal to the agent's current depth **plus** the depth step class data component this%distance because the agent is intended to deepen down.

8.3.3.29.2.3 Calculate expected food increments at the target depth

Create a virtual expected food item First, create a subjective representation of the expected food item that is used as a major reference for calculating fake override perceptions. First, calculate the fake coordinates for the expected food item, a spatial object of the class the_environment::spatial. They are equal to those of the actor agent, with the depth coordinate equal to the target depth.
Make an expected food item using the food_item standard method make (the_environment::food_item::make()) with the following parameters: the above spatial location, the size equal to the expected food gain from do_↩this, iid is set to commondata::unknown. Note that the size of the food item is reverse-calculated using the the_environment::mass2size_food() function.
Calculate the expected probability of capture (normally using the average distance to the food items under the agent perception::food_dist_below(), see above). Note that the illumination level in the calculation backend is set from the food item's current depth, i.e. the target depth of the agent. This means that the subjective illumination level used in the calculation of the capture probability is reduced automatically according to the agent's target depth.

Calculate food increments Build the expected food gain perception. The mass increment that this_agent gets from consuming this food item is defined by the_body::condition::food_fitting.

Note

Note that the_body::condition::food_fitting already subtracts processing cost automatically. Note that the expected food increment is weighted by the expected probability of capture of the expected food item.

Stomach increment from food is equal to the above value of the expected mass increment. However, stomach increment can only be zero or a positive value.

8.3.3.29.2.4 Build the fake perceptions

Body mass and stomach contents Finally, the fake perceptions for the body mass and stomach content are calculated as the current body mass minus the cost of moving to the target depth plus the expected food increment. The expected fake perception value for the stomach content at the target depth is obtained similarly by adding the expected stomach increment to the current stomach content of the agent.
The expected energy reserves perceived are calculated from the fake perceptions of the mass and length using the_body::energy_reserve() function.

Conspecifics The fake perception value for the conspecifics at the target depth is calculated directly from the this class data component this%expected_consp_number.

Predators The fake perception value for the predation risk at the target depth is calculated directly from the this class data component

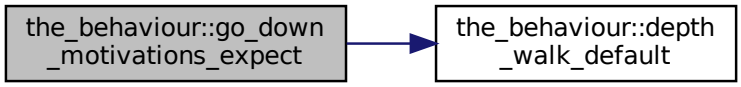
Environmental perceptions The number of food items (direct food perception) is equal to the number of food items currently under the agent.
Depth perception is according to the absolute target depth value.
Light perception is according to the new depth.

8.3.3.29.2.5 Calculate motivation expectancies The next step is to calculate the motivational expectancies using the fake perceptions to override the default (actual agent's) values. At this stage, first, calculate motivation values resulting from the behaviour done (`go_down_depth::do_this()`) at the previous steps: what would be the motivation values *if* the agent does perform GO_DOWN_DEPTH? Technically, this is done by calling the **neuronal response function**, `percept_components_motiv::motivation_components()` method, for each of the motivational states with `perception_override_` dummy parameters overriding the default values. Here is the list of the fake overriding perceptions for the GO_DOWN_DEPTH behaviour:

- `perception_override_light`
- `perception_override_depth`
- `perception_override_food_dir`
- `perception_override_predator`
- `perception_override_stomach`
- `perception_override_bodymass`
- `perception_override_energy`

Real agent perception components are now substituted by the *fake* values resulting from executing this behaviour (`reproduce::do_this()` => `the_behaviour::reproduce_do_this()` method). This is repeated for all the motivations: *hunger*, *passive avoidance, active avoidance* etc. These optional **override parameters** are substituted by the "fake" values.
Next, from the perceptual components calculated at the previous step we can obtain the **primary** and **final motivation** values by weighed summing.
Here we can use global maximum motivation across all behaviours and perceptual components if it is provided, for rescaling.
Or can rescale using local maximum value for this behaviour only.
Transfer attention weights from the actor agent `this_agent` to the `this` behaviour component. So, we will now use the updated modulated attention weights of the agent rather than their default parameter values.
So the primary motivation values are calculated.
Primary motivations are logged in the debug mode.
There is **no modulation** at this stage, so the final motivation values are the same as primary motivations.

8.3.3.29.2.6 Calculate motivation expectancies Finally, calculate the finally **expected arousal level for this behaviour**. As in the GOS, the overall arousal is the maximum value among all motivation components.
Log also the final expectancy value in the debug mode.
Now as we know the expected arousal, we can choose the behaviour which would minimise this arousal level.
Definition at line 4972 of file m_behav.f90.
Here is the call graph for this function:



8.3.3.30 go_down_do_execute()

```
subroutine the_behaviour::go_down_do_execute (
             class(go_down_depth), intent(inout) this,
```

```
class(appraisal), intent(inout) this_agent,
real(srp), intent(in), optional max_depth,
class(environment), dimension(:), intent(in), optional environments,
real(srp), intent(in), optional depth_walk )
```

Execute this behaviour component "go down" by `this_agent` agent.

Note

The "do"-function does not change the state of the this_agent or the the environment (the food item), the "execute" function **does**.

Parameters

`in,out`	*this*	[inout] this the object itself.
`in,out`	*this_agent*	[in] this_agent is the actor agent which goes down.
`in`	*max_depth*	max_depth is the optional maximum limit on the depth.
`in`	*environments*	environments optional array of the all available environments where the this agent can be in, needed for the calculation of the depth limits. If such an array of the environments is provided, `max_depth` has precedence.
`in`	*depth_walk*	depth_walk Optional downward walk size, by how deep the agent goes down.

8.3.3.30.1 Implementation details

8.3.3.30.1.1 Initial checks First, check if the size of the downward walk `depth_walk` dummy parameter is provided.
If it is not provided, it is set equal to the agent's body length multiplied by the commondata::up_down_walk_step_stdlength_factor factor parameter. Calculated by `the_behaviour::depth_walk_default()`.
Check downward step size. Here, first, check if the target depth is likely to go beyond the environment depth limits and reduce the downward walk step size accordingly. Either the explicitly provided maximum depth dummy parameter `max_depth` or an array of possible environment objects where the `this_agent` actor agent can be located is used to get the depth limit.
If the array of possible environment objects that can contain the max_depth actor agent is provided, the check involves the `the_environment::spatial::find_environment()` function to find the specific environment object the agent is currently in followed by `the_environment::environment::depth_max()` to find the minimum depth in this environment object.
If the array of possible environment objects that can contain the actor agent is not provided, the current environment is obtained from the global array the_environment::global_habitats_available. In this case, the environment that actor agent is within is determined using the the_environment::spatial::find_environment() method, which is in followed by the_environment::environment::depth_max()` to find the minimum depth in this environment object.
If `max_depth` is provided, it has precedence over the depth detected explicitly or implicitly from the environment objects.
In the case neither of the above optional parameters are provided, the maximum depth is set as the depth of the actor agent (with an additional condition that it should exceed zero), so movement down would be **impossible**.

8.3.3.30.1.2 Step 1: do_this First, we use the intent-in **do**-procedure `go_down_depth::do_this()` to perform the behaviour desired and get the **expectations of fake perceptions** for GOS. As a result, we now get this%decrement_mass_cost that defines the cost of buoyancy-based movement downwards.

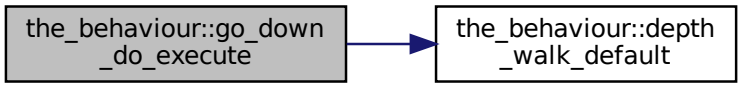
Note

At this stage, the state of the actor agent is not changed.

8.3.3.30.1.3 Step 2: Change the agent Change the location of the actor agent, moving it down to the distance this%distance.
Decrement the body mass as a consequence of transfer down. This body mass decrement constitutes the (small) energetic cost of locomotion. Call `the_body::condition::set_mass()` for this.

Additionally, also call the the_body::condition::set_length() method to update the body length history stack. However, the value_set parameter here is just the current value. This fake re-setting of the body length is done to keep both mass and length synchronised in their history stack arrays (there is no procedure for only updating history).
After resetting the body mass, update energy reserves of the agent, that depend on both the length and the mass. Check if the agent is starved to death. If yes, the agent can die without going any further.

8.3.3.30.1.4 Step 3: Change the environment Moving down by the agent does not affect the environmental objects.
Definition at line 5516 of file m_behav.f90.
Here is the call graph for this function:



8.3.3.31 go_up_depth_init_zero()

```
elemental subroutine the_behaviour::go_up_depth_init_zero (
            class(go_up_depth), intent(inout) this )
```

Initialise the **go up to a shallower spatial layer** behaviour component to a zero state.
First init components from the base root class the_behaviour::behaviour_base. Mandatory label component that should be read-only.
The execution status is always FALSE, can be reset to TRUE only when the behaviour unit is called to execution.
And the *expectancy* components.
Abstract MOVE component.
Then init components of this specific behaviour component extended class.
Definition at line 5645 of file m_behav.f90.

8.3.3.32 go_up_do_this()

```
subroutine the_behaviour::go_up_do_this (
            class(go_up_depth), intent(inout) this,
            class(appraisal), intent(in) this_agent,
            real(srp), intent(in) min_depth,
            real(srp), intent(in), optional depth_walk,
            integer, intent(in), optional predict_window_food,
            integer, intent(in), optional time_step_model )
```

Do go up by this_agent (the actor agent). Subjective assessment of the motivational value for this is based on the number of food items, conspecifics and predators at the layers below the this_agent actor agent.

Parameters

in,out	*this*	[inout] this the object itself.
in	*this_agent*	this_agent is the actor agent which goes up.
in	*min_depth*	min_depth is the maximum limit on the depth.
in	*depth_walk*	depth_walk Optional downward walk size, by how deep the agent goes down.

Parameters

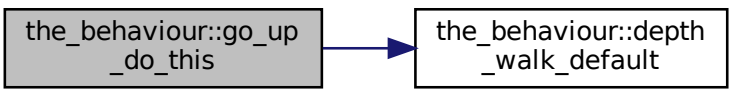
in	*predict_window_food*	predict_window_food the size of the prediction window, i.e. how many steps back in memory are used to calculate the predicted food gain. This parameter is limited by the maximum commondata::history_size_perception value of the perception memory history size.
in	*time_step_model*	time_step_model optional time step of the model, overrides the value calculated from the spatial data.

8.3.3.32.1 Implementation details First, check if the size of the upward walk `depth_walk` dummy parameter is provided.
If it is not provided, it is set equal to the agent's body length multiplied by the commondata::up_down_walk_step_stdlength_factor factor parameter. Calculated by `the_behaviour::depth_walk_default()`.
Check optional parameter for the food perception memory window. If the `predict_window_food` dummy parameter is not provided, its default value is the proportion of the whole perceptual memory window defined by commondata::history_perception_window_food. Thus, only the latest part of the memory is used for the prediction of the future food gain.
Check optional time step parameter. If unset, use global `commondata::global_time_step_model_current`.

8.3.3.32.1.1 Upward step size Here, first, check if the target depth is likely to go beyond the environment depth limits and reduce the upwnward walk step size accordingly. Namely, if the depth coordinate of the actor agent minus the depth step exceeds the minimum depth, the step is reduced to be within the available environment↩ : $d_a - D_{min} - \varepsilon$, where D_{min} is the maximum depth, d_a is the agent's current depth and ε is a very small constant defined by the parameter commondata::zero.
The upward step size component of the class is then equal to the `depth_walk`.

8.3.3.32.1.2 The cost of swimming up The expected cost of the swimming up by the buoyancy is much smaller than active propulsion. It is set as a fraction, defined by the parameter commondata::swimming_cost_factor_buoyancy_down, of active laminar propulsion calculated by function the_body::condition_cost_swimming_burst().

8.3.3.32.1.3 Calculate expected perceptions Calculate the number of conspecifics upwards of the agent using the function perception::consp_below().
Calculate the expected predation risk above the agent.
Calculate the expected food gain as an average mass of the food items above the agent. It is used by calling perception::food_mass_below() function. This expected food gain is then weighted by the subjective probability of food item capture that is calculated based on the memory the_neurobio::perception::food_probability_capture_subjective().
Definition at line 5676 of file m_behav.f90.
Here is the call graph for this function:



8.3.3.33 go_up_motivations_expect()

```
subroutine the_behaviour::go_up_motivations_expect (
             class(go_up_depth), intent(inout) this,
```

```
            class(appraisal), intent(in) this_agent,
            real(srp), intent(in), optional depth_walk,
            real(srp), intent(in), optional min_depth,
            class(environment), dimension(:), intent(in), optional environments,
            integer, intent(in), optional time_step_model,
            real(srp), intent(in), optional rescale_max_motivation )
```

go_up_depth::motivations_expect() is a subroutine (re)calculating motivations from fake expected perceptions following from the procedure go_up_depth::do_this() => the_behaviour::go_up_do_this().

Parameters

in,out	*this*	[inout] this the object itself.
in	*this_agent*	this_agent is the actor agent which goes up.
in	*depth_walk*	depth_walk The upward walk size, by how deep the agent goes up.
in	*min_depth*	min_depth is the optional maximum limit on the depth.
in	*environments*	environments optional array of the all available environments where the this agent can be in, needed for the calculation of the depth limits. If such an array of the environments is provided, min_depth` has precedence.
in	*time_step_model*	[in] time_step_model optional time step of the model, **overrides** the value calculated from the spatial data.
in	*rescale_max_motivation*	rescale_max_motivation maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

Target depth, i.e. the absolute depth of the agent after it moves up.

8.3.3.33.1 Notable local variables

8.3.3.33.1.1 Perception overrides

- **expect_food_perc_override** is the fake perception for the food items at the target depth.

expect_depth_perc_override is the fake perception of the depth, identical to the target depth.

- **expect_light_perc_override** is the fake perception of the illumination level at the target depth.
- **expect_mass_perc_override** is the fake perception value for the mass from the expected food.
- **expect_stomach_perc_override** is the fake perception value for the stomach increment from the expected food.
- **expect_energy_perc_override** is the fake perception for the energy reserves from the expected food at the target depth.
- **expected_probability_capture** is the expected probability of capture of the expected food item at the target depth.
- **expect_conspecicifc_perc_override** is the fake perception value for the number of conspecifics at the target depth.
- **expect_predator_perc_override** is fake perception value for the predation risk at the target depth.

8.3.3.33.2 Implementation details

8.3.3.33.2.1 Sanity checks and preparations Initially, check if the size of the upward walk `depth_walk` dummy parameter is provided.
If it is not provided, it is set equal to the agent's body length multiplied by the commondata::up_down_walk_step_stdlength_factor factor parameter.Calculated by `the_behaviour::depth_walk_default()`.
Check upward step size. Here, first, check if the target depth is likely to go beyond the environment depth limits and reduce the upward walk step size accordingly. Either the explicitly provided minimum depth dummy parameter `min_depth` or an array of possible environment objects where the `this_agent` actor agent can be located is used to get the depth limit.
If the array of possible environment objects that can contain the actor agent is provided, the check involves the `the_environment::spatial::find_environment()` function to find the specific environment object the agent is currently in followed by `the_environment::environment::depth_min()` to find the minimum depth in this environment object.
If the array of possible environment objects that can contain the actor agent is not provided, the current environment is obtained from the global array the_environment::global_habitats_available. In this case, the environment that actor agent is within is determined using the the_environment::spatial::find_environment() method, which is in followed by the_environment::environment::depth_max()` to find the minimum depth in this environment object.
If `min_depth` is provided, it has precedence over the depth detected from environment objects.
In the case the minimum depth cannot be determined,it is set as the depth of the actor agent (with an additional condition that it should exceed zero), so movement up would be **impossible**. Notably, it is not set to zero, a logical choice, to avoid possible asymmetric effects as the counterpart "move down" procedures use the agent's current depth as a last resort in the analogous case of no depth parameters.
If the depth coordinate of the actor agent minus the depth step is smaller than the minimum depth, the step is reduced to be strictly within the available environment. However, it should also never be below zero.
Check optional time step parameter. If not provided, use global parameter value from commondata::global_time_step_model_current.
Assess the number of food items above using the perception::food_items_above() method.
Calculate the expected distance to the food item. It is equal to the average distance to the food items perceived above in case there are any such items perceived above. Calculated using the perception::food_dist_above() method.
However, if there are no food items above (resulting a commondata::missing distance, see perception::food_dist_below()), the expected distance is set the upward walk distance `depth_walk`, that should be sufficiently long to assure the probability of food item capture is very small or zero.min_depth

8.3.3.33.2.2 Call do_this As the first step, we use the **do**-procedure `go_up_depth::do_this()` => `the_behaviour::go_up_do_this()` to perform the behaviour desired without changing either the agent or its environment, obtaining the **subjective** values of the `this` behaviour components that later feed into the motivation **expectancy** functions:

- `perception_override_light`
- `perception_override_depth`
- `perception_override_food_dir`
- `perception_override_predator`
- `perception_override_stomach`
- `perception_override_bodymass`
- `perception_override_energy`

The absolute value of the target depth is equal to the agent's current depth **minus** the depth step class data component this%distance because the agent is intended to lift up.

8.3.3.33.2.3 Calculate expected food increments at the target depth

Create a virtual expected food item First, create a subjective representation of the expected food item that is used as a major reference for calculating fake override perceptions. First, calculate the fake coordinates for the expected food item, a spatial object of the class the_environment::spatial. They are equal to those of the actor agent, with the depth coordinate equal to the target depth.

Make an expected food item using the food_item standard method `make` (the_environment::food_item::make()) with the following parameters: the above spatial location, the size equal to the expected food gain from `do_this`, iid is set to commondata::unknown. Note that the size of the food item is reverse-calculated using the the_environment::mass2size_food() function.
Calculate the expected probability of capture (normally using the average distance to the food items above the agent perception::food_dist_above()). Note that the illumination level in the calculation backend is set from the food item's current depth, i.e. the target depth of the agent. This means that the subjective illumination level used in the calculation of the capture probability is increased automatically according to the agent's target depth.

Calculate food increments Build the expected food gain perception. The mass increment that this_agent gets from consuming this food item is defined by `the_body::condition::food_fitting`.

Note

Note that `the_body::condition::food_fitting` already subtracts processing cost automatically. Note that the expected food increment is weighted by the expected probability of capture of the expected food item.

Stomach increment from food is equal to the above value of the expected mass increment. However, stomach increment can only be zero or a positive value.

8.3.3.33.2.4 Build the fake perceptions

Body mass and stomach contents Finally, the fake perceptions for the body mass and stomach content are calculated as the current body mass minus the cost of moving to the target depth plus the expected food increment. The expected fake perception value for the stomach content at the target depth is obtained similarly by adding the expected stomach increment to the current stomach content of the agent.
The expected energy reserves perceived are calculated from the fake perceptions of the mass and length using the_body::energy_reserve() function.

Conspecifics The fake perception value for the conspecifics at the target depth is calculated directly from the `this` class data component this%expected_consp_number.

Predators The fake perception value for the predation risk at the target depth is calculated directly from the `this` class data component

Environmental perceptions The number of food items (direct food perception) is equal to the number of food items currently above the agent.
Depth perception is according to the absolute target depth value.
Light perception is according to the new depth.

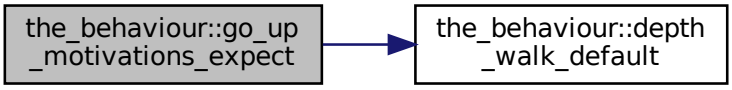
8.3.3.33.2.5 Calculate motivation expectancies The next step is to calculate the motivational expectancies using the fake perceptions to override the default (actual agent's) values. At this stage, first, calculate motivation values resulting from the behaviour done (`go_up_depth::do_this()`) at the previous steps: what would be the motivation values *if* the agent does perform GO_UP_DEPTH? Technically, this is done by calling the **neuronal response function**, `percept_components_motiv::motivation_components()` method, for each of the motivational states with `perception_override_` dummy parameters overriding the default values. Here is the list of the fake overriding perceptions for the GO_UP_DEPTH behaviour:

- `perception_override_light`
- `perception_override_depth`
- `perception_override_food_dir`
- `perception_override_predator`
- `perception_override_stomach`
- `perception_override_bodymass`

- perception_override_energy

Real agent perception components are now substituted by the *fake* values resulting from executing this behaviour (reproduce::do_this() => the_behaviour::reproduce_do_this() method). This is repeated for all the motivations: *hunger*, *passive avoidance, active avoidance* etc. These optional **override parameters** are substituted by the "fake" values.
Next, from the perceptual components calculated at the previous step we can obtain the **primary** and **final motivation** values by weighed summing.
Here we can use global maximum motivation across all behaviours and perceptual components if it is provided, for rescaling.
Or can rescale using local maximum value for this behaviour only.
Transfer attention weights from the actor agent this_agent to the this behaviour component. So, we will now use the updated modulated attention weights of the agent rather than their default parameter values.
So the primary motivation values are calculated.
Primary motivations are logged in the debug mode.
There is **no modulation** at this stage, so the final motivation values are the same as primary motivations.

8.3.3.33.2.6 Calculate motivation expectancies Finally, calculate the finally **expected arousal level for this behaviour**. As in the GOS, the overall arousal is the maximum value among all motivation components.
Log also the final expectancy value in the debug mode.
Now as we know the expected arousal, we can choose the behaviour which would minimise this arousal level.
Definition at line 5806 of file m_behav.f90.
Here is the call graph for this function:



8.3.3.34 go_up_do_execute()

```
subroutine the_behaviour::go_up_do_execute (
             class(go_up_depth), intent(inout) this,
             class(appraisal), intent(inout) this_agent,
             real(srp), intent(in), optional min_depth,
             class(environment), dimension(:), intent(in), optional environments,
             real(srp), intent(in), optional depth_walk )
```

Execute this behaviour component "go up" by this_agent agent towards.

Note

The "do"-function does not change the state of the this_agent or the the environment (the food item), the "execute" function **does**.

Parameters

in,out	*this*	[inout] this the object itself.
in,out	*this_agent*	[in] this_agent is the actor agent which goes up.
in	*min_depth*	min_depth is the optional minimum limit on the depth.
in	*environments*	environments optional array of the all available environments where the this agent can be in, needed for the calculation of the depth limits. If such an array of the environments is provided, min_depth has precedence.

Parameters

in	*depth_walk*	depth_walk Optional upward walk size, by how deep the agent goes up.

8.3.3.34.1 Implementation details

8.3.3.34.1.1 Initial checks First, check if the size of the upward walk `depth_walk` dummy parameter is provided.
If it is not provided, it is set equal to the agent's body length multiplied by the commondata::up_down_walk_step_stdlength_factor factor parameter. Calculated by `the_behaviour::depth_walk_default()`.
Check upward step size. Here, first, check if the target depth is likely to go beyond the environment depth limits and reduce the upward walk step size accordingly. Either the explicitly provided minimum depth dummy parameter `min_depth` or an array of possible environment objects where the `this_agent` actor agent can be located is used to get the depth limit.
If the array of possible environment objects that can contain the actor agent is provided, the check involves the `the_environment::spatial::find_environment()` function to find the specific environment object the agent is currently in followed by in this `the_environment::environment::depth_min()` to find the minimum depth in this environment object.
If the array of possible environment objects that can contain the actor agent is not provided, the current environment is obtained from the global array the_environment::global_habitats_available. In this case, the environment that actor agent is within is determined using the the_environment::spatial::find_environment() method, which is in followed by the_environment::environment::depth_max()` to find the minimum depth in this environment object.
If `min_depth` is provided, it has precedence over the depth detected explicitly or implicitly from the environment objects.
In the case neither of the above optional parameters are provided, the minimum depth is set as the depth of the actor agent (with an additional condition that it should exceed zero), so movement up would be **impossible**. Notably, it is not set to zero, a logical choice, to avoid possible asymmetric effects as the counterpart "move down" procedures use the agent's current depth as a last resort in the analogous case of no depth parameters.

8.3.3.34.1.2 Step 1: do_this First, we use the intent-in **do**-procedure `go_up_depth::do_this()` to perform the behaviour desired and get the **expectations of fake perceptions** for GOS. As a result, we now get this%decrement_mass_cost that defines the cost of buoyancy-based movement upwards.

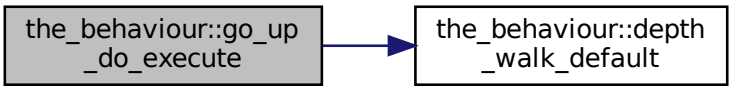
Note

At this stage, the state of the actor agent is not changed.

8.3.3.34.1.3 Step 2: Change the agent Change the location of the actor agent, moving it up to the distance this%distance.
Decrement the body mass as a consequence of transfer upwards. This body mass decrement constitutes the (small) energetic cost of locomotion. Call `the_body::condition::set_mass()` for this.
Additionally, also call the `the_body::condition::set_length()` method to update the body length history stack. However, the value_set parameter here is just the current value. This fake re-setting of the body length is done to keep both mass and length synchronised in their history stack arrays (there is no procedure for only updating history).
After resetting the body mass, update energy reserves of the agent, that depend on both the length and the mass.
Check if the agent is starved to death. If yes, the agent can die without going any further.

8.3.3.34.1.4 Step 3: Change the environment Moving down by the agent does not affect the environmental objects.
Definition at line 6351 of file m_behav.f90.

Here is the call graph for this function:



8.3.3.35 debug_base_init_zero()

```
elemental subroutine the_behaviour::debug_base_init_zero (
            class(debug_base), intent(inout) this )
```

Initialise the **fake debug behaviour** behaviour component to a zero state.
First init components from the base root class `the_neurobio::behaviour_base`. Mandatory label component that should be read-only.
The execution status is always FALSE, can be reset to TRUE only when the behaviour unit is called to execution.

Note

Note that this behaviour unit is never executed.

And the *expectancy* components.
Definition at line 6483 of file m_behav.f90.

8.3.3.36 debug_base_motivations_expect()

```
subroutine the_behaviour::debug_base_motivations_expect (
            class(debug_base), intent(inout) this,
            class(appraisal), intent(in) this_agent,
            integer, intent(in), optional time_step_model,
            real(srp), intent(in), optional rescale_max_motivation )
```

`the_behaviour::debug_base::motivations_expect()` is a subroutine (re)calculating motivations from fake expected perceptions for the **fake debug behaviour**.

Parameters

`in,out`	*this*	[inout] this the self object.
`in`	*this_agent*	this_agent is the actor agent which does reproduce.
`in`	*time_step_model*	[in] time_step_model optional time step of the model, **overrides** the value calculated from the spatial data.
`in`	*rescale_max_motivation*	rescale_max_motivation maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

8.3.3.36.1 Implementation notes

8.3.3.36.1.1 Check optional parameters Check optional time step parameter. If not provided, use global parameter value from `commondata::global_time_step_model_current`.

8.3.3.36.1.2 Main processing steps This is the **fake debug behaviour**, for which the **do**-procedure is absent.
The motivation values resulting from the behaviour are calculated for unchanged perceptions. That is, no fake perceptions are placed into the percept_components_motiv::motivation_components() procedures.
From the perceptual components calculated at the previous step we can obtain the **primary** and **final motivation** values by weighed summing.
Here we can use global maximum motivation across all behaviours and perceptual components if it is provided, for rescaling.
Or can rescale using local maximum value for this behaviour only.
Transfer attention weights from the actor agent `this_agent` to the `this` behaviour component. So, we will now use the updated modulated attention weights of the agent rather than their default parameter values.
So the primary motivation values are calculated.
Primary motivations are logged in the debug mode.
There is **no modulation** at this stage, so the final motivation values are the same as primary motivations. TODO↩ : Should include developmental or other modulation? If yes, need to separate genetic modulation component from `motivation_modulation_genetic` into a procedure bound to `MOTIVATIONS` with `this_agent` as actor.
Fourth, Calculate the finally **expected arousal level for this behaviour**. As in the GOS, the overall arousal is the maximum value among all motivation components.
Log also the final expectancy value in the debug mode.
Now as we know the expected arousal, we can choose the behaviour which would minimise this arousal level.
Definition at line 6505 of file m_behav.f90.

8.3.3.37 eat_food_item_do_this()

```
subroutine the_behaviour::eat_food_item_do_this (
             class(eat_food), intent(inout) this,
             class(appraisal), intent(in) this_agent,
             class(food_item), intent(in) food_item_eaten,
             integer, intent(in), optional time_step_model,
             real(srp), intent(in), optional distance_food_item,
             real(srp), intent(in), optional capture_prob,
             logical, intent(out), optional is_captured )
```

Eat a food item defined by the object `food_item_eaten`. The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here `the_behaviour::eat_food`). So, here the result of this procedure is assessment of the stomach content increment and body mass increment that would result from eating the **this** food item by the **this_agent**. The **main output** from this **do** procedure is the `this` behavioural unit, namely two of its internal data components:

- this%mass_increment_from_food
- this%stomach_increment_from_food

Note

The "do"-function does not change the state of the this_agent or the the environment (the food item), the "execute" function does change them.

Use subroutine rather than function as the "do"-action can potentially have several results / outputs, affect several components of the behaviour object.

There are three optional parameters which can be used as "fake" parameters in calculating fake values for subjective expectancy: `distance_food_item`, `capture_prob`, `time_step_model`. If they are not set, true objective values are calculated or used, e.g. time step of the model is taken from `commondata::global_time_step_model_current` and the distance between the agent and the food item `distance_food_item` is calculated from their spatial data.

Parameters

in,out	*this*	[inout] this the object itself.
in	*this_agent*	this_agent is the actor agent which eats the food item.
in	*food_item_eaten*	food_item_eaten is the food object that is eaten.
in	*time_step_model*	time_step_model optional time step of the model, overrides the value calculated from the spatial data.
in	*distance_food_item*	distance_food_item is the optional distance to the food item.
in	*capture_prob*	capture_prob is optional probability of capture of this food item, overrides the value calculated from the spatial data.
out	*is_captured*	is_captured optional capture flag, TRUE if the food item is captured by the agent.

8.3.3.37.1 Implementation details

8.3.3.37.1.1 Preliminary checks This food item, if found in the perception object, should be available. If not, something wrong has occurred. We cannot process an food item that has been already eaten, so no increments are done and error is reported into the log.
Check optional time step parameter.
Check distance to the food item. If provided, use the override value, if not, calculate from the the agent and the food item spatial data.
Check if food item capture probability is supplied.

Note

> If capture probability is supplied as a dummy parameter to this procedure, it will override the intrinsic capture probability that is based on the distance between the predator agent and the food item it is about to eat. This may be for example necessary when a subjective expected motivational expectancy is calculated, it can assume 100% probability and/or weightings of the resulting motivation value(s).

If the food item capture probability is not supplied, **calculate** it based on the current distance between the predator agent and this food item. (`commondata::food_item_capture_probability` is a baseline value at near-zero distance).

8.3.3.37.1.2 Processing The probability that the food item is captured is stochastic and is normally below 100%. However while calculating the behaviour expectancies, the capture probability is set to 1.0 to make the internal subjective processing deterministic. Stochastic capture success is now determined by the `the_environment::food_item::capture_success()` function.

Note

> The distance to the food item `distance_food_item_here` is used here not only to calculate the probability of food item capture (above), but also the fast burst swimming cost of approaching the food item that is about to be eaten.

Food item is captured The food item **is captured**, set the optional logical flag first.
The mass increment that this_agent gets from consuming this food item is defined by `the_body::condition::food_fitting`

Note

> Note that `the_body::condition::food_fitting` already subtracts processing cost.

Food item is not captured The food item is **not** captured, set the optional logical flag first.
If the food item is **not captured**, the agent has only to pay the energetic **processing cost** without food gain. The cost (mass decrement) is defined by `the_body::condition::food_process_cost()`. The stomach contents mass does not change in this case.
Definition at line 6725 of file m_behav.f90.

8.3.3.38 eat_food_item_motivations_expect()

```
subroutine the_behaviour::eat_food_item_motivations_expect (
            class(eat_food), intent(inout) this,
            class(appraisal), intent(in) this_agent,
            class(food_item), intent(in) food_item_eaten,
            integer, intent(in), optional time_step_model,
            real(srp), intent(in), optional distance_food_item,
            real(srp), intent(in), optional capture_prob,
            real(srp), intent(in), optional rescale_max_motivation )
```

eat_food::motivations_expect() is a subroutine (re)calculating motivations from fake expected perceptions following from the procedure eat_food::do_this() => the_behaviour::eat_food_item_do_this().

Parameters

in,out	*this*	[inout] this the self object.
in	*this_agent*	this_agent is the actor agent which does eat.
in	*food_item_eaten*	food_item_eaten is the food item object that is eaten.
in	*time_step_model*	[in] time_step_model optional time step of the model, **overrides** the value calculated from the spatial data.
in	*distance_food_item*	distance_food_item optional distance to the food item, **overrides** the value calculated from the spatial data.
in	*capture_prob*	capture_prob is optional probability of capture of this food item, **overrides** the value calculated from the spatial data.
in	*rescale_max_motivation*	rescale_max_motivation maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

8.3.3.38.1 Notable local parameters

8.3.3.38.1.1 food_capture_prob FOOD_CAPTURE_PROB is the expected (subjective) food item capture probability parameter. We assume that the agent assumes 100% probability of capture of the food item.

Note

The probability is here > 1.0 to make sure the procedure is never stochastic (subjective prob=1) and the food item is always caught (the stochastic function it is based on random_value[0..1] $<$ P).

8.3.3.38.1.2 Stomach contents stomach_increment_from_food_perc is expected increment of the stomach contents that is used in the fake perception value in the neuronal response function.
stomach_overrride_perc is the fake perception value for the stomach contents that goes into the neuronal response function.

8.3.3.38.1.3 Body mass mass_increment_from_food_perc is the expected increment of the agent's body mass that is used in the fake perception value in the neuronal response function.
bodymass_override_perc is the fake perception value for the body mass that goes into the neuronal response function.

8.3.3.38.1.4 energy_override_perc energy_override_perc is the fake perception value that goes into the neuronal response function.

8.3.3.38.1.5 capture_prob_intrinsic capture_prob_intrinsic is the intrinsic probability of capture of the this food item. It is calculated using the food_item::capture_probability() method.

8.3.3.38.2 Implementation details

8.3.3.38.2.1 Preliminary steps and checks Check optional time step parameter. If not provided, use global parameter value from `commondata::global_time_step_model_current`.
Check distance to the food item. If provided, use the override value, if not, calculate from the the agent and the food item spatial data.
Check if food item capture probability is supplied. If capture probability is supplied as a dummy parameter to this procedure, it will override the intrinsic capture probability that is based on the distance between the predator agent and the food item it is about to eat. This may be for example necessary when a subjective expected motivational expectancy is calculated, it can assume 100% probability and/or weightings of the resulting motivation value(s).
If the food item capture probability is not supplied, expectancy is based on a 100% capture probability.

Warning

> Unlike the `eat_food::do_this()` procedure where the capture probability is calculated from the true objective values, the subjective expectancies are based by default on **100% expected probability** of this food item capture.

The intrinsic (objective) probability of capture of this food item `capture_prob_intrinsic` is calculated using the `food_item::capture_probability()` method.

8.3.3.38.2.2 Main processing steps **First,** we use the **do**-procedure `eat_food::do_this()` => `the_behaviour::eat_food_item_do_this()` to perform the behaviour desired without changing either the agent or its environment and here find **representation** values that later feed into the motivation **expectancy** functions.

Note

> Note that the optional capture success flag is not used here as what is important for expectancy calculation is the agent's weight and stomach increments only.
>
> The dummy parameter `time_step_model` is not used here for calculating the capture probability because a fixed fake value of the later `FOOD_CAPTURE_PROB` is used.

We then weight the subjective increments of the body mass and stomach content that are expected from eating this food item by the **intrinsic objective capture probability** `capture_prob_intrinsic` calculated for the current time step on the basis of the distance between the agent and the food item.
After this, it is possible to calculate the fake perceptions for the stomach contents (`stomach_overrride_perc`), body mass (`bodymass_override_perc`) and the energy reserves (`energy_override_perc`). These values are ready to be passed to the neuronal response function.
Second, we calculate motivation values resulting from the behaviour done (`eat_food::do_this()`) at the previous step: what would be the motivation values *if* the agent eats this food item? This is done by calling the **neuronal response function**, `percept_components_motiv::motivation_components()` method, for each of the motivational states with `perception_override_` dummy parameters overriding the default values:

- `perception_override_stomach`;
- `perception_override_bodymass`;
- `perception_override_energy`.

Real agent perception components are now substituted by the *fake* values resulting from executing this behaviour (`eat_food::do_this()` => `the_behaviour::eat_food_item_do_this()` method). This is repeated for all the motivatios: *hunger*, *passive avoidance, active avoidance* etc. These optional **override parameters** are substituted by the "fake" values:

- `perception_override_stomach`;
- `perception_override_bodymass`;
- `perception_override_energy`.

Third, From the perceptual components calculated at the previous step we can obtain the **primary** and **final motivation** values by weighed summing.

Here we can use global maximum motivation across all behaviours and perceptual components if it is provided, for rescaling.
Or can rescale using local maximum value for this behaviour only.
Transfer attention weights from the actor agent `this_agent` to the `this` behaviour component. So, we will now use the updated modulated attention weights of the agent rather than their default parameter values.
So the primary motivation values are calculated.
Primary motivations are logged in the debug mode.
There is **no modulation** at this stage, so the final motivation values are the same as primary motivations. TODO↩ : Should include developmental or other modulation? If yes, need to separate genetic modulation component from `motivation_modulation_genetic` into a procedure bound to `MOTIVATIONS` with `this_agent` as actor.
Fourth, Calculate the finally **expected arousal level for this behaviour**. As in the GOS, the overall arousal is the maximum value among all motivation components.
Log also the final expectancy value in the debug mode.
Now as we know the expected arousal, we can choose the behaviour which would minimise this arousal level.
Definition at line 6847 of file m_behav.f90.

8.3.3.39 eat_food_item_do_execute()

```
subroutine the_behaviour::eat_food_item_do_execute (
             class(eat_food), intent(inout) this,
             class(appraisal), intent(inout) this_agent,
             class(food_item), intent(inout) food_item_eaten,
             class(food_resource), intent(inout) food_resource_real,
             logical, intent(out), optional eat_is_success )
```

Execute this behaviour component "eat food item" by `this_agent` agent towards the `food_item_eaten`.

Note

The "do"-function does not change the state of the this_agent or the the environment (the food item), the "execute" function **does** change them.

Parameters

`in,out`	*this*	[inout] this the self object.
`in,out`	*this_agent*	[inout] this_agent is the actor agent which eats the food item.
`in,out`	*food_item_eaten*	[inout] food_item_eaten is the food item object that is eaten.
`in,out`	*food_resource_real*	[inout] food_resource_real The food resource we are eating the food item in.

Note

We need to provide the food resource that the agent has perceived the food items (using the `see_food` method) because the food perception object contains **copies** of food items from the physical resource. So we have to change the availability status of the real physical resource items, not just items in the perception object of the agent.

Parameters

`out`	*eat_is_success*	eat_is_success logical indicator showing if the food item has actually been eaten (TRUE) or failed (FALSE).

8.3.3.39.1 Implementation details First, check if this food item is **not eaten** and this agent is **not dead**. It should normally the case. If not, may point to a bug.
Now process the food item by `this_agent`.

8.3.3.39.1.1 Step 1: do_this First, we use the intent-in do-procedure `eat_food::do_this()` => `the_behaviour::eat_food_item_do_this()` to perform the behaviour desired and get the **expectations of fake perceptions** for GOS. As a result, we get `mass_increment_from_food` and `stomach_↩ increment_from_food`.

Note

At this stage, the state of the food item is not changed. Only the state of `this` behaviour changes, and it will be later passed to modify the agent.

`capture_prob` is not set here, so it is set to the true objective value that depends on the distance between the predator agent and the food item, see `capture_probability` function bound to the `FOOD_ITEM` class.

Also, here set the optional output argument `eat_is_success` from the stochastic result (success/failed) of the foor item capture.
Also log the fake perceptions along with the agent's sex if running in the DEBUG mode.

8.3.3.39.1.2 Step 2: Change the agent Second, **change the agent's state** as a consequence of eating. (1) Grow the **body length** of the agent based on the mass increment from food.

Warning

Note that we increment the body length first, before incrementing/growing the body **mass**. This is because the body length increment uses the ratio of the food gain mass to the agent's body mass. So incrementing the body mass itself with the food gain should be done after the length is processed, otherwise a wrong (mass+gain) value is used.

(2). Grow the **body mass** of the agent.

Note

Note that `mass_increment_from_food` already has the processing cost subtracted. Specifically, the mass increment can be negative if the agent did not catch the food item.

Note that even if `is_captured` is False, we do call the mass and stomach increment procedures as in such a case there is a mass cost that is still subtracted (increment negative), and stomach increment is zero.

(3). And increment the **stomach contents** of the agent using `condition::stomach_increment()`.
(4). Update the energy reserves using the new currently updated mass and length by calling `condition::energy_update()`.
(5). Check if the agent is starved to death. If yes, the agent can die without going any further.

8.3.3.39.1.3 Step 3: Change the environment Third, **change the state of the environment**. Disable the food item if it is eaten and not available any more. If the capture success if False, the item is not affected.
Set the eaten status to the food item in the perception object.
We also have to set the food item in the real food resource the same eaten/absent status because the perception object may operate on a **copy of the real food objects**. So we here first get the ID number of the food item.
Second, set the food item in the food resource with the same iid the absent/eaten status.
Log the food item eaten in the debug mode.

Warning

This would result in huge amount of log writing that significantly slows down execution!

Definition at line 7248 of file m_behav.f90.

8.3.3.40 reproduce_init_zero()

```
elemental subroutine the_behaviour::reproduce_init_zero (
            class(reproduce), intent(inout) this )
```

Initialise reproduce behaviour object.
First init components from the base root class `the_behaviour::behaviour_base`: Mandatory label component that should be read-only.

The execution status is always FALSE, can be reset to TRUE only when the behaviour unit is called to execution.
And the *expectancy* type components.
And init the expected arousal data component.
Second, init components of this specific behaviour (REPRODUCE) component extended class.

Note

Note that we initialise increments to 0.0, not MISSING as increments will be later added. And several items can be added consecutively.

Definition at line 7450 of file m_behav.f90.

8.3.3.41 maximum_n_reproductions()

```
integer function the_behaviour::maximum_n_reproductions (
            class(appraisal), intent(in) this )
```

Calculate the maximum number of possible reproductions for this agent. It is assumed that a male can potentially fertilise several females that are within its perception object (in proximity) during a single reproduction event. For females, this number if always one.

Returns

The maximum number of reproductions (successful fertilisations) within the same reproduction event.

Initialise the number of same- and opposite-sex conspecifics (integer counters) to zero.

8.3.3.41.1 Implementation details **First,** determine if there are any conspecifics in the perception, if there are no, reproduction is impossible. Return straight away with zero result in such a case.
Second, check if this agent is **female**. If yes, only one fertilisation is possible, so return max_num=1.
Exit from the procedure afterwards.
From now on, it is assumed the agent is male. **Third,** determine how many conspecific male agents in the perception object have testosterone level **higher** than this actor agent. These conspecific male agents can take part in the fertilisation. However, all male conspecifics with testosterone **lower** than in this agent are out-competed by this agent and the other high-testosterone males and would not be involved in reproduction.
Finally, calculate the expected number of fertilised females, i.e. the number of reproductions for this agent assuming only this agent and all other male agents with the testosterone levels exceeding that in this agent can reproduce.
Definition at line 7485 of file m_behav.f90.

8.3.3.42 reproduce_do_this()

```
subroutine the_behaviour::reproduce_do_this (
            class(reproduce), intent(inout) this,
            class(appraisal), intent(in) this_agent,
            real(srp), intent(in), optional p_reproduction,
            logical, intent(out), optional is_reproduce )
```

Do reproduce by this_agent (the actor agent) given the specific probability of successful reproduction. The probability of reproduction depends on the number of agents of the same and of the opposite sex within the visual range of the this agent weighted by the difference in the body mass between the actor agent and the average body mass of the other same-sex agents. The **main output** from this **do** procedure is the this behavioural unit object, namely its two components:

- this%reprfact_decrement_testosterone
- this%reprfact_decrement_estrogen

Parameters

in,out	*this*	[inout] this the object itself.
in	*this_agent*	this_agent is the actor agent which does/does not reproduce.

Parameters

in	*p_reproduction*	p_reproduction optional probability of reproduction, overrides the value calculated from this_agent data.
out	*is_reproduce*	is_reproduce optional reproduction success flag, TRUE if the reproduction is successfully done by the agent.

8.3.3.42.1 Implementation details Determine if the agent's hormonal system is ready for reproduction, that its current level of sex steroids σ_i exceeds the baseline (initially determined by the genome) σ_0 by a factor ν determined by the parameter commondata::sex_steroids_reproduction_threshold:

$$\sigma_i > \nu\sigma_0.$$

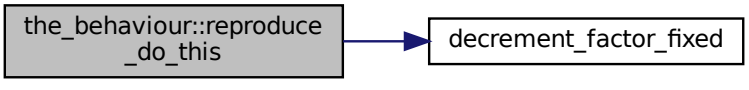
This check is done by the the_body::is_ready_reproduce() function.

- If the level of sex steroids is insufficient, reproduction is impossible and the values of gonadal steroid decrements get are zero. The reproduction indicator is_reproduce if present, is also set to FALSE and no further processing is then performed.

Determine if there are any conspecifics in the perception, if there are no, reproduction is impossible. Return straight away with zero values of gonadal steroid decrements, as in the case of unsuccessful reproduction. The reproduction indicator is_reproduce if present, is also set to FALSE.
Check optional probability of reproduction dummy parameter. If it is absent, use the value calculated from the this_agent agent's perception data calling probability_reproduction() method. This is the **upper limit** on the reproduction probability provided the actor agent has sufficient motivation and resources.
Then we call stochastic logical function reproduction_success() to determine the **actual outcome of reproduction**.
If reproduction is **successful**, the reproductive factor gonadal steroid (hormonal) components reproduce::reprfact_decrement_testoste and reproduce::reprfact_decrement_estrogen data component are determined in sex specific manner:

- in males testosterone is decreased,
- in females, estrogen is decreased.

An additional condition is that the level of the gonadal hormones should not fall below the baseline level. Additionally, the cost of reproduction, body mass decrement reproduce::decrement_mass, is calculated and set using the reproduction::reproduction_cost() method.
Also, if is_reproduce optional parameter is provided, set it to TRUE.
If reproduction is **not successful**, reproduction factor decrements equal to zero are returned. The body mass decrement is equivalent to the reproduction cost of unsuccessful reproduction (reproduction::reproduction_cost_unsucces
Additionally, set is_reproduce to FALSE if it is provided.
Definition at line 7561 of file m_behav.f90.
Here is the call graph for this function:



8.3.3.43 reproduce_motivations_expect()

```
subroutine the_behaviour::reproduce_motivations_expect (
            class(reproduce), intent(inout) this,
            class(appraisal), intent(in) this_agent,
            integer, intent(in), optional time_step_model,
            real(srp), intent(in), optional reprod_prob,
            logical, intent(in), optional non_stochastic,
            real(srp), intent(in), optional rescale_max_motivation )
```

`reproduce::motivations_expect()` is a subroutine (re)calculating motivations from fake expected perceptions following from `reproduce::do_this()` => `the_behaviour::reproduce_do_this()` procedure.

Parameters

`in,out`	*this*	[inout] this the self object.
`in`	*this_agent*	this_agent is the actor agent which does reproduce.
`in`	*time_step_model*	[in] time_step_model optional time step of the model, **overrides** the value calculated from the spatial data.
`in`	*reprod_prob*	reprod_prob is optional probability of reproduction for the this actor agent, **overrides** the value calculated from the agent data using the probability_reproduction() function.
`in`	*non_stochastic*	non_stochastic is a logical flag that sets 100% probability of reproduction. This parameter has precedence over the `reprod_prob`.
`in`	*rescale_max_motivation*	rescale_max_motivation maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

8.3.3.43.1 Notable local parameters

8.3.3.43.1.1 probability_reproduction_base_def `PROBABILITY_REPRODUCTION_BASE_DEF` is the expected (subjective) probability of reproduction; set as a parameter.
We assume that the agent assumes 100% probability of reproduction.

Note

The probability is here > 1.0 to make sure the procedure is never stochastic (subjective prob=1) and reproduction always performed (it is based on random_value[0..1] $<$ P).

8.3.3.43.1.2 reproduction_prob_intrinsic `reproduction_prob_intrinsic` is the probability of reproduction that is intrinsic for the agent at the given conditions, calculated using the probability_reproduction() function.

8.3.3.43.1.3 reprfactor_percept `reprfactor_percept` is the value of the reproductive factor that goes as a fake perception value into the neuronal response function. This reproductive factor is determined in a sex specific way:

- `reprfact_decrement_testosterone` in males;
- `reprfact_decrement_estrogen` in females.

8.3.3.43.1.4 body_mass_percept `body_mass_percept` is the "subjective" value of the energetic cost of reproduction that goes as a fake perception value into the neuronal response function. It is calculated via the `reproduction::reproduction_cost()` method.

8.3.3.43.1.5 energy_override_perc `energy_override_perc` is the fake perception value for the energy reserves that goes into the neuronal response function.

8.3.3.43.2 Implementation details First, calculate the intrinsic probability of reproduction for this actor agent using the probability_reproduction() method.

8.3.3.43.2.1 Check optional parameters Check optional time step parameter. If not provided, use global parameter value from commondata::global_time_step_model_current.
Check if the probability of reproduction is supplied. If the probability of reproduction is supplied as a dummy parameter to this procedure, it will override the intrinsic probability of reproduction for this actor agent that is calculated using the probability_reproduction() method. This may be for example necessary when a subjective motivational expectancy is calculated, it can assume 100% probability and/or weightings of the resulting motivation value(s).
If the probability of reproduction is not supplied, expectancy is based on the intrinsic probability_reproduction() value.
If the non_stochastic dummy parameter is set to TRUE, the probability of reproduction is obtained from the PROBABILITY_REPRODUCTION_BASE_DEF local parameter that is 1.1. In such a case, it guarantees that the agent will always reproduce.

Note

Unlike the reproduce::do_this() procedure where the reproduction probability is calculated from the true objective values, the subjective expectancies are based by default on **100% expected probability** of this agent reproduction.

8.3.3.43.2.2 Main processing steps **First,** we use the **do**-procedure reproduce::do_this() => the_behaviour::reproduce_do_this() to perform the behaviour desired without changing either the agent or its environment, obtaining the **subjective** values of the this behaviour components that later feed into the motivation **expectancy** functions:

- reprfact_decrement_testosterone
- reprfact_decrement_estrogen

We then weight the expected subjective decrements of the reproductive factor components of he_neurobio↩ ::reproduce class, testosterone or estrogen, that are intrinsically expected for the actor agent by the **objective probability of reproduction** reproduction_prob_intrinsic (calculated for the current time step using the **intrinsic** the_neurobio::probability_reproduction() method). The reproductive factor reprfactor_↩ percept that goes into the neuronal response function as a fake perception is based on gonadal steroid (hormonal) components: reprfact_decrement_testosterone and reprfact_decrement_estrogen in a sex specific manner:

- in males testosterone is weighted by reproduction_prob_intrinsic,
- in females, estrogen is weighted by reproduction_prob_intrinsic.

The same is done for the subjective assessment of the body mass cost of reproduction (body_mass_percept): it is weighted by the intrinsic probability of reproduction (reproduction_prob_intrinsic).
At this point, therefore, the fake perception values for the reproductive factor (reprfactor_percept) and body mass (body_mass_percept) are known. Finally, calculate also the fake perception for the energy reserves (energy_override_perc) using the the_body::energy_reserve() procedure.
Second, we calculate motivation values resulting from the behaviour done (reproduce::do_this() => the_behaviour::reproduce_do_this()) at the previous step: what would be the motivation values *if* the agent doe perform reproduction? Technically, this is done by calling the **neuronal response function**, percept_components_motiv::motivation_components() method, for each of the motivational states with perception_override_ dummy parameters overriding the default values: perception↩ _override_reprfac and also perception_override_energy.
Real agent perception components are now substituted by the *fake* values resulting from executing this behaviour (reproduce::do_this() => the_behaviour::reproduce_do_this() method). This is repeated for all the motivations: *hunger*, *passive avoidance, active avoidance* etc. These optional **override parameters** are substituted by the "fake" values:

- perception_override_reprfac;
- perception_override_bodymass.

Third, From the perceptual components calculated at the previous step we can obtain the **primary** and **final motivation** values by weighed summing.
Here we can use global maximum motivation across all behaviours and perceptual components if it is provided, for rescaling.
Or can rescale using local maximum value for this behaviour only.
Transfer attention weights from the actor agent `this_agent` to the `this` behaviour component. So, we will now use the updated modulated attention weights of the agent rather than their default parameter values.
So the primary motivation values are calculated.
Primary motivations are logged in the debug mode.
There is **no modulation** at this stage, so the final motivation values are the same as primary motivations. TODO↩ : Should include developmental or other modulation? If yes, need to separate genetic modulation component from `motivation_modulation_genetic` into a procedure bound to `MOTIVATIONS` with `this_agent` as actor.
Fourth, Calculate the finally **expected arousal level for this behaviour**. As in the GOS, the overall arousal is the maximum value among all motivation components.
Log also the final expectancy value in the debug mode.
Now as we know the expected arousal, we can choose the behaviour which would minimise this arousal level.
Definition at line 7692 of file m_behav.f90.

8.3.3.44 reproduce_do_execute()

```
subroutine the_behaviour::reproduce_do_execute (
             class(reproduce), intent(inout) this,
             class(appraisal), intent(inout) this_agent )
```

Execute this behaviour component "reproduce" by the `this_agent` agent.

Parameters

`in,out`	*this_agent*	[inout] this_agent is the actor agent which reproduces.

8.3.3.44.0.1 Notable local variables *body_mass_after* is the updated body mass of the agent excluding the cost of reproduction.

8.3.3.44.1 Implementation details

8.3.3.44.1.1 Basic checks First, check if there are **any conspecifics** in the perception object of the agent and this agent is **not dead**. It should normally the case. If not, may point to a bug.

8.3.3.44.1.2 Check the agent condition Calculate the updated body mass of the agent after reproduction `body_mass_after`. It is obtained by subtracting the cost of reproduction from the current body mass of the agent. The cost of reproduction is calculated using the function `reproduction::reproduction_cost()` (=> `the_body::reproduction_cost_energy()`). Therefore it does not necessarily coincide with the subjective cost of reproduction that is kept in the `the_behaviour::reproduce` class.
Additionally, check if the energy reserves of the agent and the body mass are enough for reproduction. That is, if the agent survives following the reproduction and does not get starved to death. The check is done using the `the_body::is_starved()` function in the named if block `CHECK_STARVED_AFTER`.

- If the condition of the agent is insufficient for reproduction, it is assumed that the agent **has attempted** reproduction but was not successful. Then, the `reproduction_unsuccessful_cost_subtract()` procedure is called to subtract some small cost if unsuccessful reproduction.
- Following this, exit and **return** back from this procedure.

8.3.3.44.1.3 Step 1: do_this First, we use the intent-in do-procedure `reproduce::do_this()` => `the_behaviour::reproduce_do_this()` to perform the behaviour desired and get the **expectations of fake perceptions** for GOS:

- this%reprfact_decrement_testosterone
- this%reprfact_decrement_estrogen.

At this stage, the state of the agent is not changed. Only the state of `this` behaviour changes, and it will be later passed to modify the agent. The `do_this` procedure also returns the stochastic status of the reproduction event `is_reproduce` is TRUE if the reproduction event was successful.
Also log the fake perceptions along with the agent's sex if running in the DEBUG mode.

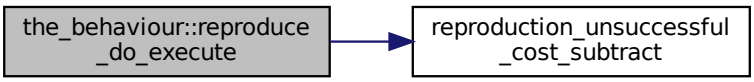
8.3.3.44.1.4 Step 2: Change the agent

Check reproduction success Second, **change the agent's state** as a consequence of reproduction. Check if reproduction event was stochastically successful. If the reproduction event was not successful (`is_reproduce` is FALSE), the `reproduction_unsuccessful_cost_subtract()` procedure is called to subtract some small cost of unsuccessful reproduction.
Following this, exit and **return** back from this procedure.

Process reproducing agent From now on it is assumed that the reproduction event was stochastically **successful**. (A) Update the number of successful reproductions and the number of offspring that result from this reproduction, for this agent, by the default number (=1) calling `reproduction::reproductions_increment()`.
(B) Decrease the sex steroids level following the reproduction. This is different in males and females: testosterone is decreased in males and estrogen, in females. An additional condition is that the level of gonadal steroids could not fall to less than the baseline.
(C) Decrement the body mass as a consequence of reproduction. This body mass decrement constitutes the energetic cost of reproduction. The updated body mass (after subtraction of the cost) has already been calculated as `body_mass_after`.
Additionally, also call the `the_body::condition::set_length()` method to update the body length history stack. However, the value_set parameter here is just the current value. This fake re-setting of the body length is done to keep both mass and length synchronised in their history stack arrays (there is no procedure for only updating history).
After resetting the body mass, update energy reserves of the agent, that depend on both the length and the mass.
(D). Check if the agent is starved to death. If yes, the agent can die without going any further.

8.3.3.44.1.5 Step 3: Change the environment

Reproduction of the agent does not affect the environmental objects. TODO: add method to do actual reproduction crossover mate choice and produce eggs
Definition at line 8069 of file m_behav.f90.
Here is the call graph for this function:



8.3.3.45 walk_random_do_this()

```
subroutine the_behaviour::walk_random_do_this (
             class(walk_random), intent(inout) this,
             class(appraisal), intent(in) this_agent,
             real(srp), intent(in), optional distance,
             real(srp), intent(in), optional distance_cv,
             integer, intent(in), optional predict_window_pred,
```

```
integer, intent(in), optional predict_window_food,
integer, intent(in), optional time_step_model )
```

The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here WALK_RANDOM).

Parameters

in	*this_agent*	this_agent is the actor agent which eats the food item.
in	*distance*	distance is an optional walk distance. If stochastic Gaussian walk is set, this value defines the average distance.

Note

Even though the walk distance is internally defined in terms of the agent's body length, this parameter defines the **absolute distance** in cm.

Parameters

in	*distance_cv*	distance_cv is an optional coefficient of variation for the random walk distance. If absent, non-stochastic walk step size is used.
in	*predict_window_pred*	predict_window_pred the size of the prediction window, i.e. how many steps back in memory are used to calculate the predicted general predation risk. This parameter is limited by the maximum commondata::history_size_perception value of the perception memory history size.
in	*predict_window_food*	predict_window_food the size of the prediction window, i.e. how many steps back in memory are used to calculate the predicted food gain. This parameter is limited by the maximum commondata::history_size_perception value of the perception memory history size.
in	*time_step_model*	time_step_model optional time step of the model, overrides the value calculated from the spatial data.

8.3.3.45.1 Implementation details

8.3.3.45.1.1 Checks and preparations Check optional time step parameter. If unset, use global commondata::global_time_step_model_current.
Check optional parameter for the general predation risk perception memory window. If the predict_window_pred dummy parameter is not provided, its default value is the proportion of the whole perceptual memory window defined by commondata::history_perception_window_pred. Thus, only the latest part of the memory is used for the prediction of the future predation risk.
Check optional parameter for the food perception memory window. If the predict_window_food dummy parameter is not provided, its default value is the proportion of the whole perceptual memory window defined by commondata::history_perception_window_food. Thus, only the latest part of the memory is used for the prediction of the future food gain.

8.3.3.45.1.2 Calculate the distance of swimming The normal locomotion distance is fixed to the fraction of the agent current body length set by the parameter commondata::walk_random_distance_default_factor. This is a baseline value that can serve as the mean in case of stochastic walks (the_environment::spatial_moving::rwalk()) or as the actual value in case of deterministic walks.
The walk distance $D_{rw} = L\varrho$ where L is the agent's body length and ϱ is the parameter factor commondata::walk_random_distance_default_factor.
However, if the walk distance is provided as an optional parameter distance to this procedure, this provided value is used as the baseline distance instead. This allows to easily implement several types of walks, e.g. "long" (migration-like) and short (local).
This baseline distance value D_{rw} is saved into the this behaviour data component %distance.

- If the `distance_cv` optional dummy parameter is set to a non-zero value (> commondata::tolerance_high_def_srp), the the walk distance is *stochastic* with the mean equal to the above baseline value and the coefficient of variation set by the %distance_cv data component of the `this` walk object, that is in turn equal to the `distance_cv` parameter.

The %distance is then reset to a Gaussian value, creating an error/uncertainty in the expectancy.

- If `distance_cv` parameter is absent or is explicitly set to zero, the walk distance is deterministic with the value equal to the baseline. Also, the %distance_c data component is 0.0 for non-stochastic distances.

This allows to implement uncertainty in the walk distance depending on different factors, such as the arousal or hormone level.

8.3.3.45.1.3 Calculate expected cost of the swimming The expected cost of swimming in the random walk depends on the walk distance and is calculated using the the_body::condition::cost_swim() assuming *laminar* flow (laminar flow is due to normal relatively slow swimming pattern).

8.3.3.45.1.4 Calculate expected food item perception *Food item* perception expected after a random walk is calculated using the the_behaviour::hope() function mechanism.
First, average number of food items in the "older" and "newer" parts of the memory is calculated using the the_neurobio::memory_perceptual::get_food_mean_n_split() procedure. (Note that the `split_val` parameter to this procedure is not provided so the default 1/2 split is used.)
Second, the expected number of food items following the walk (%expected_food_dir) is calculated based on the the_behaviour::hope() function mechanism. Here, the baseline value f_0 is the current number of food items in the food perception object of the actor agent, and the historical ratio ϱ is calculated as the mean number of food items in the "older" to "newer" memory parts:

$$\varrho = \frac{\overline{n_2}}{\overline{n_1}}.$$

The grid arrays for the hope function are defined by the obtained from commondata::walk_random_food_hope_abscissa and commondata::walk_random_food_hope_ordinate parameter arrays.

8.3.3.45.1.5 Calculate expected food gain The expected food gain is calculated differently depending on the mean distance of the random walk.

- If the agent currently has any food items in perception, the short walk is defined as a walk with the distance not exceeding commondata::walk_random_food_gain_hope units of the average to the food items in perception.
- If there is no food in the perception object, a walk is "short" if it does not exceed commondata::walk_random_food_gain_hope_ag units of the agent body length.

Short walks For relatively short walks, the expected food gain is based on the currently available value.
The expected food gain is equal to the average mass of the food item in the latest `predict_window_food_↩here` steps of the memory stack, weighted by the average number of food items in the same width latest memory if this number is less than 1 or 1 (i.e. unweighted) if their number is higher.

$$\begin{cases} F_{exp} = \overline{f(m)} \cdot \overline{n(m)}, & \overline{n(m)} < 1; \\ F_{exp} = \overline{f(m)}, & \overline{n(m)} \geq 1 \end{cases}$$

where $\overline{f(m)}$ is the average mass of the food items and $\overline{n(m)}$ is the average number of food items in the m latest steps of the perceptual memory stack.
The averages are calculated with the_neurobio::memory_perceptual::get_food_mean_size() and the_neurobio::memory_perceptual::g
The average mass of the food items is calculated from their average size using the the_environment::size2mass_food() function.
Thus, if the agent had previously some relatively poor perceptual history of encountering food items, so that the average number of food items is fractional < 1 (e.g. average number 0.5, meaning that it has seen a single food item approximately every other time step), the expected food is weighted by this fraction (0.5). If, on the other hand,

the agent had several food items at each time step previously, the average food item size is unweighted (weight=1.↩
0). This conditional weighting reflects the fact that it is not possible to eat more than one food item at a time in this model version.
This expected food gain is then weighted by the subjective probability of food item capture that is calculated based on the memory the_neurobio::perception::food_probability_capture_subjective().

Long walks For relatively long walks, the expected food gain is based on the the_behaviour::hope() function.
First, average size of food items in the "older" and "newer" parts of the memory is calculated using the the_neurobio::memory_perceptual::get_food_mean_size_split() procedure. (Note that the `split_val` parameter to this procedure is not provided so the default 1/2 split is used.)
Second, the values of the "old" and "new" *food gain* used to calculate the expectations are obtained by weighting the respective average mass of the food item by the average number of food items if this number is less than 1 or 1 (i.e. unweighted) if their average number is higher.

$$\begin{cases} f_1 = \overline{m_1} \cdot \overline{n_1}, & \overline{n_1} < 1 \\ f_1 = \overline{m_1}, & \overline{n_1} \geq 1 \end{cases} \qquad \begin{cases} f_2 = \overline{m_2} \cdot \overline{n_2}, & \overline{n_2} < 1 \\ f_2 = \overline{m_2}, & \overline{n_2} \geq 1 \end{cases}$$

where $\overline{m_1}$ is the average mass of the food items and $\overline{n_1}$ is the average number of food items in the "older" half of the perceptual memory stack and $\overline{m_2}$ is the average mass of the food items and $\overline{n_2}$ is the average number of food items in the "newer" half of the memory stack.
Thus, if the agent had some relatively poor perceptual history of encountering food items, so that the average *number* of food items is fractional < 1 (e.g. average number 0.5, meaning that it has seen a single food item approximately every other time step), the food gain is weighted by this fraction (0.5). If, on the other hand, the agent had more than one food items at each time step previously, the average food item size is unweighted (weight=1.0). This conditional weighting reflects the fact that it is not possible to eat more than one food item at a time in this model version.

Note

A similar expectancy assessment mechanism is used in the assessment of the food gain expectancy for the the_behaviour::migrate behaviour component the_behaviour::migrate_do_this().

The next step is to calculate the baseline food gain f_0, against which the expectancy based on the the_behaviour::hope() function is evaluated. This baseline value is obtained by weighting the average mass of the food items in the whole memory stack $\overline{m}$ by their average number $\overline{n}$ provided this number is *n<1* as above:

$$\begin{cases} f_0 = \overline{m} \cdot \overline{n}, & \overline{n} < 1; \\ f_0 = \overline{m}, & \overline{n} \geq 1 \end{cases}$$

This baseline food gain is then weighted by the subjective probability of food item capture that is calculated based on values from the the memory the_neurobio::perception::food_probability_capture_subjective().
Finally, the the_behaviour::hope() function is called with the above estimates for the baseline food gain, its "older" and "newer" values. The *zero hope ratio* and the *maximum hope* parameters are obtained from commondata::migrate_food_gain_ratio_zero_hope and commondata::migrate_food_gain_maximum_hope parameter constants.

8.3.3.45.1.6 Calculate expected predation risk The expected risk of predation (as the food gain above) is calculated differently for relatively short and long walks. The walk is considered *short* if the distance does not exceed commondata::walk_random_pred_risk_hope_agentl units of the agent body lengths and *long* otherwise.
First, the current level of the direct risk of predation is calculated using the_neurobio::perception::risk_pred() based on the perception of the `this_agent` agent assuming the agent is not freezing (because it is going to move a random walk).
Second, calculate the current value of the general predation risk using the the_neurobio::predation_risk_backend() procedure. The size of this limited memory window is set by the `predict_window_pred` dummy parameter.

Note

In contrast to the above limited prediction memory window, calculation of the predation risk in the "objective" procedure the_neurobio::perception_predation_risk_objective() uses the same backend but the *whole* memory window (commondata::history_size_perception).

Short walk In short walks, the expected values are just equal to the above current direct extimates.

- **Direct** risk of predation is equal to the current value as calculated above using the_neurobio::perception::risk_pred().
- **General** risk, the expected *general* risk of predation in random walk is equal to the current value of direct predation risk as above.

Long walk On the other hand, for long walks, the expected values of the risks are based on the the_behaviour::hope() function mechanism.

- First, calculate the older and newer predation averages from the memory stack using the the_neurobio::memory_perceptual::get method. These averages serve as the base point for calculating the new to old ratio in the the_behaviour::hope() function.
- The **direct risk** of predation is based on the_behaviour::hope() function. If the number of predators in the latest memory (predation risk) is increasing in the local environment, its expectancy in the unknown environment at a long distance diminishes, if the risk is reducing over time in the agent's perception, the expectancy increases. The hope grid values for the general predation hope function are defined by the commondata::migrate_predator_zero_hope and commondata::migrate_predator_maximum_hope parameter constants.

  Note

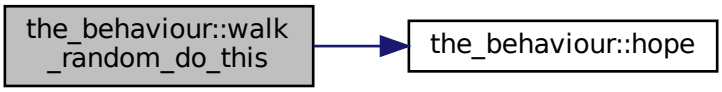
  Note that the hope function constants used here are the same as for the_behaviour::migrate.

- The expectancy value of **general predation risk** after long walk is obtained via the the_behaviour::hope() function. If the number of predators (risk) is increasing in the latest perception memory, its expectancy after long walk diminishes, if the perceived risk is reducing over time, the expectancy increases. The hope grid values for the general predation hope function are defined by the commondata::migrate_predator_zero_hope and commondata::migrate_predator_maximum_hope parameter constants.

  Note

  Note that the hope function constants used here are the same as for the_behaviour::migrate.

Definition at line 8267 of file m_behav.f90.
Here is the call graph for this function:



8.3.3.46 walk_random_motivations_expect()

```
subroutine the_behaviour::walk_random_motivations_expect (
             class(walk_random), intent(inout) this,
             class(appraisal), intent(in) this_agent,
             real(srp), intent(in), optional distance,
             real(srp), intent(in), optional distance_cv,
             integer, intent(in), optional predict_window_pred,
             integer, intent(in), optional predict_window_food,
```

```
            integer, intent(in), optional time_step_model,
            real(srp), intent(in), optional rescale_max_motivation )
```

the_behaviour::walk_random::expectancies_calculate() (re)calculates motivations from fake expected perceptions following from the procedure walk_random::do_this() => the_behaviour::walk_random_do_

Parameters

in	*this_agent*	this_agent is the actor agent which does random walk.
in	*distance*	distance is an optional walk distance. If stochastic Gaussian walk is set, this value defines the average distance.
in	*distance_cv*	distance_cv is an optional coefficient of variation for the random walk distance. If absent, non-stochastic walk step size is used.
in	*predict_window_pred*	predict_window_pred the size of the prediction window, i.e. how many steps back in memory are used to calculate the predicted general predation risk. This parameter is limited by the maximum commondata::history_size_perception value of the perception memory history size.
in	*predict_window_food*	predict_window_food the size of the prediction window, i.e. how many steps back in memory are used to calculate the predicted food gain. This parameter is limited by the maximum commondata::history_size_perception value of the perception memory history size.
in	*time_step_model*	[in] time_step_model optional time step of the model, **overrides** the value calculated from the spatial data.
in	*rescale_max_motivation*	rescale_max_motivation optional maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

The probability of capture of the expected food object.
Expected food gain that is fitting into the stomach of the agent.

8.3.3.46.1 Notable local variables

8.3.3.46.1.1 Perception overrides

- **perception_override_food_dir** is the expected number of food items in perception.

perception_override_pred_dir is the expected direct predation risk.

- **perception_override_predator** is the expected general predation risk, that is based on a weighting of the current predation and predation risk from the memory stack.
- **perception_override_stomach** is the expected stomach contents as a consequence of random walk.
- **perception_override_bodymass** is the expected body mass as a consequence of the random walk.
- **perception_override_energy** is the expected energy reserves as a consequence of the escape movement. Calculated from the body mass and weight.

8.3.3.46.2 Implementation details

8.3.3.46.2.1 Checks and preparations Check optional distance of walk. If it is absent, defined as commondata::walk_random_distance_default_factor times of the agent body length.
Check optional time step parameter. If not provided, use global parameter value from commondata::global_time_step_model_current.
Check optional parameter for the general predation risk perception memory window. If the predict_window_↩ pred dummy parameter is not provided, its default value is the proportion of the whole perceptual memory window defined by commondata::history_perception_window_pred. Thus, only the latest part of the memory is used for the prediction of the future predation risk.

Check optional parameter for the food perception memory window. If the `predict_window_food` dummy parameter is not provided, its default value is the proportion of the whole perceptual memory window defined by commondata::history_perception_window_food. Thus, only the latest part of the memory is used for the prediction of the future food gain.

8.3.3.46.2.2 Call do_this As the first step, we use the **do**-procedure walk_random::do_this() => the_behaviour::walk_random_do_ to perform the behaviour desired without changing either the agent or its environment, obtaining the **subjective** values of the `this` behaviour components that later feed into the motivation **expectancy** functions:

- `perception_override_food_dir`
- `perception_override_pred_dir`
- `perception_override_predator`
- `perception_override_stomach`
- `perception_override_bodymass`
- `perception_override_energy`

8.3.3.46.2.3 Calculate expected (fake) perceptions

Fake perception for the food items The expected perception of the number of food items that the agent is going to see following the walk is calculated in the `do_this` procedure. Here it is obtained from the %expected_food_dir data component of the class.

Fake perception of stomach content First, create a fake food item with the spatial position identical to that of the agent. The position is used only to calculate the illumination and therefore visual range. The cost(s) are calculated providing explicit separate distance parameter, so the zero distance from the agent is inconsequential. The size of the food item is obtained from the expected food gain by the reverse calculation function the_environment::mass2size_food(). Standard `make` method for the food item class is used.
Second, calculate the **probability of capture** of this expected food item. The probability of capture of the fake food item is calculated using the the_environment::food_item::capture_probability() backend assuming the distance to the food item is equal to the average distance of all food items in the **current perception** object. However, if the agent does not see any food items currently, the distance to the fake food item is assumed to be equal to the visibility range weighted by the (fractional) commondata::walk_random_dist_expect_food_uncertain_fact parameter. Thus, the expected *raw* food gain (in the `do`-function) is based on the past memory whereas the probability of capture is based on the latest perception experience.
Third, the expected food gain corrected for fitting into the agent's current stomach and capture cost is obtained by the_body::condition::food_fitting(). It is then weighted by the expected capture probability. Note that the probability of capture (weighting factor) is calculated based on the current perception (see above), but the travel cost is based on the actual expected %distance.
Stomach content: the perception override value for the stomach content is obtained incrementing the current stomach contents by the nonzero expected food gain, adjusting also for the digestion decrement (the_body::stomach_emptify_backend()).
Body mass: the **body mass** perception override is obtained by incrementing (or decrementing if the expected food gain is negative) the current body mass by the expected food gain and also subtracting the cost of living component.
Energy: The fake perception values for the energy reserves (`energy_override_perc`) using the `the_body::energy_reserve()` procedure.
Predation risk: finally, fake perceptions of predation risk are obtained from the values calculated in the `do` procedure: the_behaviour::walk_random::expected_pred_dir_risk and the_behaviour::walk_random::expected_predation_risk.

8.3.3.46.2.4 Calculate motivation expectancies The next step is to calculate the motivational expectancies using the fake perceptions to override the default (actual agent's) values. At this stage, first, calculate motivation values resulting from the behaviour done (`walk_random::do_this()`) at the previous steps: what would be the motivation values *if* the agent does perform WALK_RANDOM? Technically, this is done by calling the **neuronal response function**, `percept_components_motiv::motivation_components()` method, for each of the motivational states with `perception_override_` dummy parameters overriding the default values. Here is the list of the fake overriding perceptions for the WALK_RANDOM behaviour:

- perception_override_food_dir
- perception_override_pred_dir
- perception_override_predator
- perception_override_stomach
- perception_override_bodymass
- perception_override_energy

Real agent perception components are now substituted by the *fake* values resulting from executing this behaviour (reproduce::do_this() => the_behaviour::reproduce_do_this() method). This is repeated for all the motivations: *hunger*, *passive avoidance, active avoidance* etc. These optional **override parameters** are substituted by the "fake" values.

8.3.3.46.2.5 Calculate primary and final motivations Next, from the perceptual components calculated at the previous step we can obtain the **primary** and **final motivation** values by weighed summing.
Here we can use global maximum motivation across all behaviours and perceptual components if it is provided, for rescaling.
Or can rescale using local maximum value for this behaviour only.
Transfer attention weights from the actor agent this_agent to the this behaviour component. So, we will now use the updated modulated attention weights of the agent rather than their default parameter values.
So the primary motivation values are calculated.
Primary motivations are logged in the debug mode.
There is **no modulation** at this stage, so the final motivation values are the same as primary motivations.

8.3.3.46.2.6 Calculate motivation expectancies Finally, calculate the finally **expected arousal level for this behaviour**. As in the GOS, the overall arousal is the maximum value among all motivation components.
Log also the final expectancy value in the debug mode.
Now as we know the expected arousal, we can choose the behaviour which would minimise this arousal level.
Definition at line 8731 of file m_behav.f90.

8.3.3.47 walk_random_do_execute()

```
subroutine the_behaviour::walk_random_do_execute (
             class(walk_random), intent(inout) this,
             class(appraisal), intent(inout) this_agent,
             real(srp), intent(in), optional step_dist,
             real(srp), intent(in), optional step_cv,
             class(environment), intent(in), optional environment_limits )
```

Execute this behaviour component "random walk" by this_agent agent.

Parameters

in,out	*this_agent*	[in] this_agent is the actor agent which goes down.
in	*step_dist*	step_dist optional fixed distance of the walk. In case the coefficient of variation (next optional parameter) is provided, the walk distance is stochastic with the later coefficient of variation.
in	*step_cv*	step_cv Optional coefficient of variation for the walk step, if not provided, the step CV set by the parameter commondata::walk_random_distance_stochastic_cv.
in	*environment_limits*	environment_limits Limits of the environment area available for the random walk. The moving object cannot get beyond this limit. If this parameter is not provided, the environmental limits are obtained automatically from the global array the_environment::global_habitats_available.

8.3.3.47.1 Implementation details

8.3.3.47.1.1 Checks and preparations Check if the optional coefficient of variation for the step size. If the parameter is not provided, the CV is set from the parameter commondata::walk_random_distance_stochastic_cv.

Warning

To set deterministic walk, the coefficient of variation should be explicitly set to 0.0. This is different from the expectancy procedures , which assume deterministic default walk (CV=0.0).

8.3.3.47.1.2 Step 1: do_this First, we use the intent-in **do**-procedure the_behaviour::walk_random::do_this() to perform the behaviour desired. However, Expectancies for food gain and predator risk that are not used at this stage.

8.3.3.47.1.3 Step 2: Change the agent

Perform walk The agent does the random walk with the step size this%distance. Therefore, it is now possible to change the state of the agent.
Random walk is done in the "2.5D" mode, i.e. with separate parameters for the horizontal distance (and CV) and vertical depth distance (and its CV). This is done to avoid potentially a too large vertical displacement of the agent (vertical migration involves separate behaviours). Thus, the vertical shift distance should normally be smaller than the horizontal shift. The difference between the main horizontal and smaller vertical shifts is defined by the parameter commondata::walk_random_vertical_shift_ratio. Note that the coefficient of variation for the vertical walk component is set separately using the ratio commondata::walk_random_vertical_shift_cv_ratio.
The agent performs the random walk using the main the_environment::spatial_moving::rwalk() procedure. If the limiting environment is known (`environment_limits` optional parameter), the `rwalk` call also includes it. If environmental limits are not provided as a dummy parameter, they are obtained automatically from the global array the_environment::global_habitats_available.

Process the cost of movement

- Reset the body mass of the actor agent subtracting the actual cost of moving that is automatically calculated in the call to the_body::condition::cost_swim(). The the_body::condition::set_mass() method is used here to adjust the mass.

Additionally, also call the `the_body::condition::set_length()` method to update the body length history stack. However, the value_set parameter here is just the current value. This fake re-setting of the body length is done to keep both mass and length synchronised in their history stack arrays (there is no procedure for only updating history).

- After resetting the body mass, update energy reserves of the agent, that depend on both the length and the mass.

Finally, check if the agent is starved to death. If yes, the agent can die without going any further.

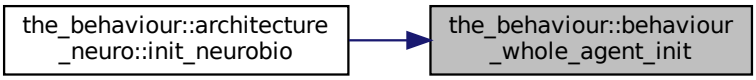
8.3.3.47.1.4 Step 3: Change the environment Random walk does not affect the environmental objects.
Definition at line 9189 of file m_behav.f90.

8.3.3.48 behaviour_whole_agent_init()

```
elemental subroutine, private the_behaviour::behaviour_whole_agent_init (
            class(behaviour), intent(inout) this )  [private]
```

Initialise the behaviour components of the agent, the the_behaviour::behaviour class.

8.3.3.48.1 Implementation notes Initialise the label for the currently executed behaviour to an easily discernible value (e.g. by `gre`).
Initialise the execution status of each of the behaviour units that compose this class to FALSE (these behavioural units are not executing).
Cleanup the history stack of behaviour labels the_behaviour::behaviour::history_behave.
Initialise anchor behav_debug_indicators "debugging indicators" for the_behaviour::behaviour class.
Definition at line 9315 of file m_behav.f90.
Here is the caller graph for this function:



8.3.3.49 behaviour_whole_agent_deactivate()

```
elemental subroutine the_behaviour::behaviour_whole_agent_deactivate (
            class(behaviour), intent(inout) this )
```

Deactivate all behaviour units that compose the behaviour repertoire of the agent.
Definition at line 9342 of file m_behav.f90.

8.3.3.50 behaviour_get_behaviour_label_executing()

```
elemental character(len=label_length) function the_behaviour::behaviour_get_behaviour_label_↩
executing (
            class(behaviour), intent(in) this )
```

Obtain the label of the currently executing behaviour for the `this` agent.
Definition at line 9361 of file m_behav.f90.

8.3.3.51 behaviour_select_conspecific()

```
integer function the_behaviour::behaviour_select_conspecific (
            class(behaviour), intent(inout) this,
            real(srp), intent(in), optional rescale_max_motivation )
```

Select the optimal conspecific among (possibly) several ones that are available in the **perception object** of the agent.

Parameters

in	*rescale_max_motivation*	rescale_max_motivation maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

Returns

The function returns the index of the food item that is chosen for eating (if there are any food items within the perception object of the agent) or 0 otherwise.

8.3.3.51.1 Notable local variables

- **n_seen_percepis** is the total number of food items found in the perception object.

expected_gos_consp is an *array* of motivational GOS expectancies from each of the food items within the perception object.

8.3.3.51.2 Implementation details

8.3.3.51.2.1 Preparation steps First, check if the agent has any conspecific(s) within its perception objects using `perception::has_consp()` method. Return zero straight away if no conspecifics are seen. Therefore, from now on it is assumed that the agent has at least one conspecific in its perception object.
The local variable `n_seen_percep` is the total number of conspecifics found in the perception object.
If there is only one conspecific, get its number (1) and exit. There is no choice if only a single conspecific is here.
Check if the maximum motivation value for rescale is provided as a parameter.
If provided, use global maximum motivation across all behaviours and perceptual components for rescaling.
If not provided, rescale using local maximum motivation value for this agent.

8.3.3.51.2.2 Calculate GOS expectancies Calculate GOS expectancies from approaching each of the conspecifics in the perception object. This is implemented in the `CONSP_EXPECT` loop.

CONSP_EXPECT loop

- First, initialise the behavioural state. Specifically, the `approach_conspec::init()` method initialises the attention weights.
- Second, calculate the motivation GOS expectancies that would result if the agent approaches to each of the conspecifics that are in its perception object. The method approach_conspec::expectancies_calculate()` does the job.

  Note

  Note that the target offset parameter is absent, which means that the default value, average body size of the agent and its target, is used. TODO: or set explicitly?

- Now we can get an array of motivational GOS expectancies from approaching each of the conspecifics within the perception object: `expected_gos_consp`.

8.3.3.51.2.3 Select minimum arousal items Once we calculated GOS motivational expectancies for all the food items, we can determine which of the food items results in the **minimum** arousal.
Definition at line 9373 of file m_behav.f90.

8.3.3.52 behaviour_select_conspecific_nearest()

```
integer function the_behaviour::behaviour_select_conspecific_nearest (
            class(behaviour), intent(in) this )
```

Select the nearest conspecific among (possibly) several ones that are available in the perception object. Note that conspecifics are sorted by distance within the perception object. Thus, this procedure just selects the first conspecific.

Returns

The function returns the index of the first conspecific if there are any within the perception object of the agent, 0 otherwise.

Definition at line 9487 of file m_behav.f90.

8.3.3.53 behaviour_select_food_item()

```
integer function the_behaviour::behaviour_select_food_item (
            class(behaviour), intent(inout) this,
            real(srp), intent(in), optional rescale_max_motivation )
```

Select the optimal food item among (possibly) several ones that are available in the **perception object** of the agent. Choosing the optimal food item to catch may be a non-trivial task and different decision rules could be implemented for this.

Parameters

in	*rescale_max_motivation*	rescale_max_motivation maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

Returns

The function returns the index of the food item that is chosen for eating (if there are any food items within the perception object of the agent) or 0 otherwise.

8.3.3.53.1 Notable local variables

- **n_seen_percepis** is the total number of food items found in the perception object.

expected_gos_fitem is an *array* of motivational GOS expectancies from each of the food items within the perception object.

8.3.3.53.2 Implementation details

8.3.3.53.2.1 Preparation steps First, check if the agent has any food item(s) within its perception objects using `perception::has_food()` method. Return zero straight away if no food seen. Therefore, from now on it is assumed that the agent has at least one food item in its perception object.
The local variable `n_seen_percep` is the total number of food items found in the perception object.
If there is only one food item, get its number (1) and exit.
Check if the maximum motivation value for rescale is provided as a parameter.
If provided, use global maximum motivation across all behaviours and perceptual components for rescaling.
If not provided, rescale using local maximum motivation value for this agent.

8.3.3.53.2.2 Calculate GOS expectancies Calculate GOS expectancies from each of the food items in the perception object. This is implemented in the `ITEMS_EXPECT` loop.

ITEMS_EXPECT loop

- First, initialise the behavioural state. Specifically, the `eat_food::init()` method initialises the attention weights.
- Second, calculate the motivation GOS expectancies from each of the food item in the perception object of the **this** agent. The `eat_food::expectancies_calculate()` does the job.
- Now we can get an array of motivational GOS expectancies from each of the food items within the perception object: `expected_gos_fitem`.

8.3.3.53.2.3 Select minimum arousal items Once we calculated GOS motivational expectancies for all the food items, we can determine which of the food items results in the **minimum** arousal.
Definition at line 9507 of file m_behav.f90.

8.3.3.54 behaviour_select_food_item_nearest()

```
integer function the_behaviour::behaviour_select_food_item_nearest (
            class(behaviour), intent(in) this )
```

Select the nearest food item among (possibly) several ones that are available in the perception object. This is a specific and **most simplistic** version of the `behaviour_select_food_item` function: select the nearest food item available in the agent's perception object. Because the food items are sorted within the perception object just select the first item.

Returns

The function returns the index of the first food item if there are any food items within the perception object of the agent or 0 otherwise.

Definition at line 9675 of file m_behav.f90.

8.3.3.55 behaviour_do_eat_food_item()

```
subroutine the_behaviour::behaviour_do_eat_food_item (
            class(behaviour), intent(inout) this,
            integer, intent(in), optional number_in_seen,
            class(food_resource), intent(inout) food_resource_real )
```

Eat a specific food item that are found in the perception object.

Parameters

in	*number_in_seen*	The index of the first food item (if there are any food items) within the perception object of the agent. If not set, default is the first (nearest) food item in the perception object.
in,out	*food_resource_real*	[inout] food_resource_real The food resource the agent is eating the food item in. Note that it could be a joined food resource composed with the_environment::join() procedure for assembling several habitats into the the_environment::global_habitats_available array or resources collapsed using the the_environment::food_resource::join() method.

8.3.3.55.1 Implementation details First, check if the agent has any food items in its perception object using the perception::has_food() method. Return straight away if no food perceived.
If there are no food items in the perception object or nothing is chosen, exit without any further processing. Normally this should not occur as the `perception::has_food()` check method guarantees that there are some food items in the perception object.
If this check is passed set the id of the food perception object.
Finally, init the behaviour object `eat` before "execute". Calls the eat_food::init() method.
Set the currently executed behaviour label. It is from the the_behaviour::behaviour_base::label data component of the base class.
Set the execution status for all behaviours to FALSE and then for this specific behaviour to TRUE. Only one behaviour unit can be executed at a time.
Do eat the food item chosen using the `execute` method of the `EAT_FOOD` class: eat_food::execute().
Update (add to stack) the agent's history of behaviours the_behaviour::behaviour::history_behave: string labels of the behaviours are are saved.
Update (increment) the agent's debugging indicators from indicators:

- individual count of the_behaviour::eat_food occasions;

individual count of successful food items %n_eaten_indicator.
Shift the position of the agent to the position of the food item eaten. That is, the agent itself moves to the spatial position that has been occupied by the food item that has just been consumed.
Definition at line 9692 of file m_behav.f90.

8.3.3.56 behaviour_do_reproduce()

```
subroutine the_behaviour::behaviour_do_reproduce (
            class(behaviour), intent(inout) this )
```

Reproduce based on the `this` agent's current state.

8.3.3.56.1 Implementation details First, check if there are any conspecifics in the perception object. Return straight away if no conspecifics seen. No cost of reproduction is subtracted in such a case.
Then, Init the behaviour (`reproduce::init()` => `the_behaviour::reproduce_init_zero`) before "execute".
Set the currently executed behaviour label. It is from the the_behaviour::behaviour_base::label data component of the base class.
Set the execution status for all behaviours to FALSE and then for this specific behaviour to TRUE. Only one behaviour unit can be executed at a time.
Finally, do the reproduction using the `reproduce::execute()` => `the_behaviour::reproduce_do_execute()` method from the `the_behaviour::reproduce` class.
Update (add to stack) the agent's history of behaviours the_behaviour::behaviour::history_behave: string labels of the behaviours are are saved.
Finally, update the positional history stack the_environment::spatial_moving::history: the current spatial position of the agent is re-saved in the history stack using the_environment::spatial_moving::repeat_position().

Note

Re-saving the current position is necessary to keep the full positional history even for the behaviours that do not involve spatial displacement (movement).

Definition at line 9784 of file m_behav.f90.

8.3.3.57 behaviour_do_walk()

```
subroutine the_behaviour::behaviour_do_walk (
            class(behaviour), intent(inout) this,
            real(srp), intent(in), optional distance,
            real(srp), intent(in), optional distance_cv )
```

Perform a random Gaussian walk to a specific average distance with certain variance (set by the CV).

Parameters

in	*distance*	distance is an optional walk distance. If stochastic Gaussian walk is set, this value defines the average distance.

Note

Even though the walk distance is internally defined in terms of the agent's body length, this parameter defines the **absolute distance** in cm.

Parameters

in	*distance_cv*	distance_cv is an optional coefficient of variation for the random walk distance. If absent, non-stochastic walk step size is used.

8.3.3.57.1 Implementation notes

- First, check if the walk distance is provided as a dummy parameter, and if not, the default value is set by the commondata::walk_random_distance_default_factor times of the agent body length.

Then check if the Coefficient of Variation of the distance parameter is also provided. If no, the default If the

distance_cv optional dummy parameter is set to the value defined by the commondata::walk_random_distance_stochastic_cv parameter.

- Initialise the behavioural component for this behaviour, the_behaviour::walk_random::init().

Set the currently executed behaviour label. It is from the the_behaviour::behaviour_base::label data component of the base class.
Set the execution status for all behaviours to FALSE and then for this specific behaviour to TRUE. Only one behaviour unit can be executed at a time.

- Finally, call the execute method for this behaviour: the_behaviour::walk_random::execute().

Update (add to stack) the agent's history of behaviours the_behaviour::behaviour::history_behave: string labels of the behaviours are are saved.
Definition at line 9831 of file m_behav.f90.

8.3.3.58 behaviour_do_freeze()

```
subroutine the_behaviour::behaviour_do_freeze (
            class(behaviour), intent(inout) this )
```

Perform (execute) the the_behaviour::freeze behaviour.

8.3.3.58.1 Implementation notes This behaviour has no parameters (e.g. target) and is rather trivial to execute:

- initialise the behaviour using the the_behaviour::freeze::init() method.

Set the currently executed behaviour label. It is from the the_behaviour::behaviour_base::label data component of the base class.
Set the execution status for all behaviours to FALSE and then for this specific behaviour to TRUE. Only one behaviour unit can be executed at a time.

- execute the behaviour with the_behaviour::freeze::execute() method.

Update (add to stack) the agent's history of behaviours the_behaviour::behaviour::history_behave: string labels of the behaviours are are saved.
Finally, update the positional history stack the_environment::spatial_moving::history: the current spatial position of the agent is re-saved in the history stack using the_environment::spatial_moving::repeat_position() method.

Note

Re-saving the current position is necessary to keep the full positional history even for the behaviours that do not involve spatial displacement (movement).

Definition at line 9900 of file m_behav.f90.

8.3.3.59 behaviour_do_escape_dart()

```
subroutine the_behaviour::behaviour_do_escape_dart (
            class(behaviour), intent(inout) this,
            class(spatial), intent(in), optional predator_object )
```

Perform (execute) the the_behaviour::escape_dart behaviour.

Parameters

in	*predator_object*	predator_object optional predator object, if present, it is assumed the actor agent tries to actively escape from this specific predator.

8.3.3.59.1 Implementation notes

- Initialise the behaviour using the the_behaviour::escape_dart::init() method.

Set the currently executed behaviour label. It is from the the_behaviour::behaviour_base::label data component of the base class.
Set the execution status for all behaviours to FALSE and then for this specific behaviour to TRUE. Only one behaviour unit can be executed at a time.

- Execute the behaviour with the_behaviour::escape_dart::execute() method. Note that if the target predator object is not provided, a default predator with the size commondata::predator_body_size is assumed. See the_behaviour::escape_dart_do_this() for details.

Update (add to stack) the agent's history of behaviours the_behaviour::behaviour::history_behave: string labels of the behaviours are are saved.
Definition at line 9942 of file m_behav.f90.

8.3.3.60 behaviour_do_approach()

```
subroutine the_behaviour::behaviour_do_approach (
             class(behaviour), intent(inout) this,
             class(spatial), intent(in) target_object,
             logical, intent(in), optional is_random,
             real(srp), intent(in), optional target_offset )
```

Approach a specific the_environment::spatial class target, i.e. execute the the_behaviour::approach behaviour. The target is either a conspecific from the perception (the_neurobio::conspec_percept_comp class) or any arbitrary the_environment::spatial class object.

Parameters

in	*target_object*	target_object is the spatial target object the actor agent is going to approach.
in	*is_random*	is_random indicator flag for random correlated walk. If present and is TRUE, the agent approaches to the `target_object` in form of random correlated walk (see the_environment::spatial_moving::corwalk()), otherwise directly.
in	*target_offset*	target_offset is an optional offset for the target, so that the target position of the approaching agent does not coincide with the target object. If absent, a default value set by the commondata::approach_offset_default is used. For the the_behaviour::approach_conspec, the default value is as an average of the agent and target conspecific body lengths.

8.3.3.60.1 Implementation details

- Check the optional parameter flag: `is_random`: if the parameter is set to TRUE, a random Gaussian walk towards the target object is done, otherwise a direct direct approach towards the target object leaving the target offset distance is performed.

Check the type of the target object. Different targets are processed differently for approach.

- If it is of the class the_neurobio::conspec_percept_comp (i.e. conspecific perception object):
  - the default target offset is set to the average body sizes of the agent and its target conspecific;
  - The the_behaviour::approach_conspec behaviour class is initialised with the the_behaviour::approach_conspec::init() method;
  - Finally, approach to the conspecific is executed with the the_behaviour::approach_conspec::execute() method.

Update (add to stack) the agent's history of behaviours the_behaviour::behaviour::history_behave: string labels of the behaviours are are saved.

- If, on the other hand, the target object is of the any other class (i.e. it is an arbitrary object):
  - The default target offset is set by the the parameter constant commondata::approach_offset_default;
  - The the_behaviour::approach_spatial behaviour class is initialised with the the_behaviour::approach_spatial::init() method;
  - Finally, approach to the conspecific is executed with the the_behaviour::approach_spatial::execute() method.

Update (add to stack) the agent's history of behaviours the_behaviour::behaviour::history_behave: string labels of the behaviours are are saved.
Definition at line 9987 of file m_behav.f90.

8.3.3.61 behaviour_do_migrate()

```
subroutine the_behaviour::behaviour_do_migrate (
             class(behaviour), intent(inout) this,
             class(environment), intent(in) target_env )
```

Perform (execute) the the_behaviour::migrate (migration) behaviour.

Parameters

in	*target_env*	target_env the target environment the actor agent is going to (e)migrate into.

8.3.3.61.1 Implementation notes

- Initialise the the_behaviour::migrate behaviour component using the the_behaviour::migrate::init() method.

Set the currently executed behaviour label. It is from the the_behaviour::behaviour_base::label data component of the base class.
Set the execution status for all behaviours to FALSE and then for this specific behaviour to TRUE. Only one behaviour unit can be executed at a time.

- The "migrate" behaviour is executed by the the_behaviour::migrate::execute() method.

Update (add to stack) the agent's history of behaviours the_behaviour::behaviour::history_behave: string labels of the behaviours are are saved.
Definition at line 10090 of file m_behav.f90.

8.3.3.62 behaviour_try_migrate_random()

```
logical function the_behaviour::behaviour_try_migrate_random (
             class(behaviour), intent(inout) this,
             class(environment), intent(in) target_env,
             real(srp), intent(in), optional max_dist,
             real(srp), intent(in), optional prob )
```

Perform a simplistic random migration. If the agent is within a specific distance to the target environment, it emigrates there with a specific fixed probability.

Parameters

in	*target_env*	target_env the target environment the actor agent is going to (e)migrate into.
in	*max_dist*	max_dist Optional maximum distance, in units of the agent's body size, towards the target environment when the agent can (probabilistically) emigrate into it.
in	*prob*	prob Probability of migration

Returns

Logical flag that shows if the agent has actually emigrated (TRUE) or not (FALSE).

8.3.3.62.1 Notable variables

- **point_target_env** is the target point inside the target environment to which this agent is going to relocate.

distance_target is the distance to the target environment.

- **MAX_DIST_DEFAULT** is the default maximum distance towards the target environment (units of the agent's body size) when the agent can emigrate into it. This default distance is set by the parameter commondata::migrate_random_max_dist_target. However, note that the migration is probabilistic and occurs with the probability `prob`.

8.3.3.62.2 Implementation notes The function returns FALSE whenever the agent has not actually migrated into the target environment.

8.3.3.62.2.1 Optional parameters Optional parameters `max_dist` and `prob` are checked and the default values are set in case any of them is absent.

- `max_dist` = `MAX_DIST_DEFAULT` (commondata::migrate_random_max_dist_target)
- `prob` = 0.5.

8.3.3.62.2.2 Calculate the distance towards the target environment First, determine the nearest target point within the target environment and calculate the distance to the target point.
The distance towards the target environment (and the target point in this environment) is defined as the minimum distance towards all segments limiting this environment in the 2D X x Y projection

Warning

This is valid only for the simple box environment implementation. Generally, it equals to the minimum distance across all the polyhedrons limiting the target environment).

The target point for the migrating agent within the target environment is then not just the edge of the target environment, but some point penetrating inside to some distance defined by the parameter commondata::migrate_dist_penetrate_offset (in units of the agent's body length). The the_environment::environment::nearest_target() method is used to find the closest point in the target environment and the (smallest) distance towards this environment, these values are adjusted automatically for the offset parameter in the procedure call.

8.3.3.62.2.3 Move to the target environment with probability "prob" If the distance towards the target environment does not exceed `max_dist` body lengths of the agent, the agent can move into this target environment, exactly to the target point `point_target_env` with the probability `prob`.

- If the agent has emigrated into the target environment, the output logical flag `is_migrated` is set to TRUE. (Otherwise, it is always FALSE.)

Process the cost of movement This only concerns the cases when the agent had migrated into the target environment `target_env`.

- Reset the body mass of the agent subtracting the actual cost of the migration moving that is automatically calculated in the call to the_body::condition::cost_swim(). The the_body::condition::set_mass() method is used here to adjust the mass.

Additionally, also call the `the_body::condition::set_length()` method to update the body length history stack. However, the value_set parameter here is just the current value. This fake re-setting of the body length is done to keep both mass and length synchronised in their history stack arrays (there is no procedure for only updating history).

- After resetting the body mass, update energy reserves of the agent, that depend on both the length and the mass.

Finally, check if the agent is starved to death. If yes, the agent can die without going any further.
Definition at line 10127 of file m_behav.f90.

8.3.3.63 behaviour_do_go_down()

```
subroutine the_behaviour::behaviour_do_go_down (
             class(behaviour), intent(inout) this,
             real(srp), intent(in), optional depth_walk )
```

Perform (execute) the the_behaviour::go_down_depth (go down) behaviour.

Parameters

in	*depth_walk*	depth_walk Optional downward walk size, by how deep the agent goes down.

8.3.3.63.1 Implementation notes

- Initialise the the_behaviour::go_down_depth behaviour component using the the_behaviour::go_down_depth::init() method.

Set the currently executed behaviour label. It is from the the_behaviour::behaviour_base::label data component of the base class.
Set the execution status for all behaviours to FALSE and then for this specific behaviour to TRUE. Only one behaviour unit can be executed at a time.

- The "go down" behaviour is executed by calling the the_behaviour::go_down_depth::execute() method. Note that the walk length can be provided by dummy parameter `depth_walk`, otherwise the default step is used that is equal to the commondata::up_down_walk_step_stdlength_factor times of the agent body length.

Update (add to stack) the agent's history of behaviours the_behaviour::behaviour::history_behave: string labels of the behaviours are are saved.
Definition at line 10267 of file m_behav.f90.

8.3.3.64 behaviour_do_go_up()

```
subroutine the_behaviour::behaviour_do_go_up (
             class(behaviour), intent(inout) this,
             real(srp), intent(in), optional depth_walk )
```

Perform (execute) the the_behaviour::go_up_depth (go up) behaviour.

Parameters

in	*depth_walk*	depth_walk Optional downward walk size, by how deep the agent goes up.

8.3.3.64.1 Implementation notes

- Initialise the the_behaviour::go_up_depth behaviour component using the the_behaviour::go_up_depth::init() method.

Set the currently executed behaviour label. It is from the the_behaviour::behaviour_base::label data component of the base class.
Set the execution status for all behaviours to FALSE and then for this specific behaviour to TRUE. Only one behaviour unit can be executed at a time.

- The "go up" behaviour is executed by calling the the_behaviour::go_up_depth::execute() method. Note that the walk length can be provided by dummy parameter `depth_walk`, otherwise the default step is used that is equal to the commondata::up_down_walk_step_stdlength_factor times of the agent body length.

Update (add to stack) the agent's history of behaviours the_behaviour::behaviour::history_behave: string labels of the behaviours are are saved.
Definition at line 10310 of file m_behav.f90.

8.3.3.65 behaviour_cleanup_history()

```
elemental subroutine the_behaviour::behaviour_cleanup_history (
            class(behaviour), intent(inout) this )
```

Cleanup the behaviour history stack for the agent. All values are empty.
Definition at line 10354 of file m_behav.f90.

8.3.3.66 behaviour_select_optimal()

```
subroutine the_behaviour::behaviour_select_optimal (
            class(behaviour), intent(inout) this,
            real(srp), intent(in), optional rescale_max_motivation,
            class(food_resource), intent(inout), optional food_resource_real )
```

Select and **execute** the optimal behaviour, i.e. the behaviour which minimizes the expected GOS arousal.

Note

Note that the "select" method should be called **after** the the_neurobio::perception, the_neurobio::appraisal and the Global Organismic State (the_neurobio::gos_global) objects were obtained.

Parameters

`in,out`	*food_resource_real*	[inout] food_resource_real The food resource the agent is eating the food item in. Note that it could be a joined food resource composed with the_environment::join() procedure for assembling several habitats into the the_environment::global_habitats_available array or resources collapsed using the the_environment::food_resource::join() method.
`in`	*rescale_max_motivation*	rescale_max_motivation maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

8.3.3.66.1 Notable local variables

- **expected_gos_debug_base** is the GOS expectancy for the fake debug behaviour unit the_behaviour::debug_base : it does not depend on any fake perceptions and represents a baseline estimate. This behaviour unit also does not participate in the procedure that selects the minimum arousal.

expected_gos_eat is the GOS expectancy value predicted from eating the optimal food item.

- **food_item_selected** is the optimal food item selected from all those that are currently within the perception object of the agent.
- **expected_gos_reproduce** is the GOS expectancy value predicted from reproduction.
- **expected_gos_walk** is the GOS expectancy value predicted from the Gaussian random walk of the optimal step size.
- **walk_distance_selected** - the static step (from values in the commondata::behav_walk_step_stdlen_static array).

- **expected_gos_freeze** is the GOS expectancy value predicted from freezing.
- **expected_gos_escape** is the GOS expectancy value predicted from escape movement.
- **predator_selected_n** - the predator object within the perception, that is associated with the lowest GOS arousal of escape, i.e. the most subjectively dangerous predator for the agent. Thus is actually the *number* of the predator within the perception object.
- **expected_gos_approach_conspec** is the GOS expectancy value predicted from the approach to conspecific behaviour.
- **conspec_selected_n** - the conspecific object within the perception, that is associated with the lowest GOS arousal of approach, i.e. the most subjectively attractive conspecific for the agent. Thus is actually the *number* of the conspecific within the perception object.
- **expected_gos_migrate** is the GOS expectancy value predicted from migration behaviour into the optimal habitat, i.e. the habitat within the array of available habitats commondata::global_habitats_available that minimises the linked GOS arousal.
- **habitat_selected_n** - the number of the habitat object within the commondata::global_habitats_available array, that is associated with the lowest GOS arousal of the migration behaviour, i.e. the most subjectively attractive habitat for the agent.
- **expected_gos_depth_down** is the GOS expectancy value predicted from the downward vertical migration with the optimal step size.
- **go_down_distance_selected** - the static step size for the downwards vertical migration (from values in the commondata::behav_go_up_down_step_stdlen_static array).
- **expected_gos_depth_up** is the GOS expectancy value predicted from the upward vertical migration with the optimal step size.
- **go_up_distance_selected** - the static step size for the upwards vertical migration (from values in the commondata::behav_go_up_down_step_stdlen_static array).
- **expected_gos_all** is the array that contains GOS arousal values for all of the behaviours that count when calculating the minimum.

Warning

Automatic allocation of the `expected_gos_all` array might not work on all compilers and platforms. If manually allocated, check the exact number of behaviour units.

8.3.3.66.2 Implementation details

8.3.3.66.2.1 Checks and preparations Determine optional parameter `rescale_max_motivation`. If it is absent from the parameter list, the value is calculated from the current perception using the the_neurobio::motivation::max_perception() method.

8.3.3.66.2.2 Calculate the motivational expectancies First, the expectancies of the GOS arousal from each of the available behaviour units are calculated.

- **Debug base fake behaviour** (the_behaviour::debug_base) calling debug_base_select(). This behaviour does not enter in the competition of behaviour units for arousal minimisation and is useful only in the debug mode.

Eat food (the_behaviour::eat_food) calling eat_food_select().

- **Reproduce** (the_behaviour::reproduce) calling reproduce_select().
- **Random walks** (the_behaviour::walk_random) calling walk_random_select().
- **Freezing** (the_behaviour::freeze) calling freeze_select().

- **Escape** (the_behaviour::escape_dart) calling escape_dart_select().
- **Approach to a spatial object** (the_behaviour::apprach): Approach to an arbitrary spatial object is not used in this version, this behaviour is never executed.
- **Approach conspecifics** (the the_behaviour::approach_conspec) calling approach_consp_select().
- **Migrate** (the_behaviour::migrate) calling migrate_select().
- **Go down** (the_behaviour::go_down_depth) calling go_down_select().
- **Go up** (the the_behaviour::go_up_depth) calling go_up_select().

8.3.3.66.2.3 Execute behaviours that minimise GOS arousal After the GOS arousal values for all behaviour units are calculated, the agent can determine the minimum value and what is the associated behaviour unit that minimises the GOS arousal.
First, an array containing all GOS arousal values for all of the above behaviour units is constructed `expected_↩ gos_all`.

Note

Note that there is no `allocate` command here as all fairly modern Fortran compilers support automatic allocation of arrays on intrinsic assignment. This feature should work by default in GNU gfortran v.4.6 and Intel ifort v.17.0.1. Automatic allocation allows to avoid a possible bug when the number of array elements in the `allocate` statement is not updated when the `expected_gos_` components of the array are added or removed.

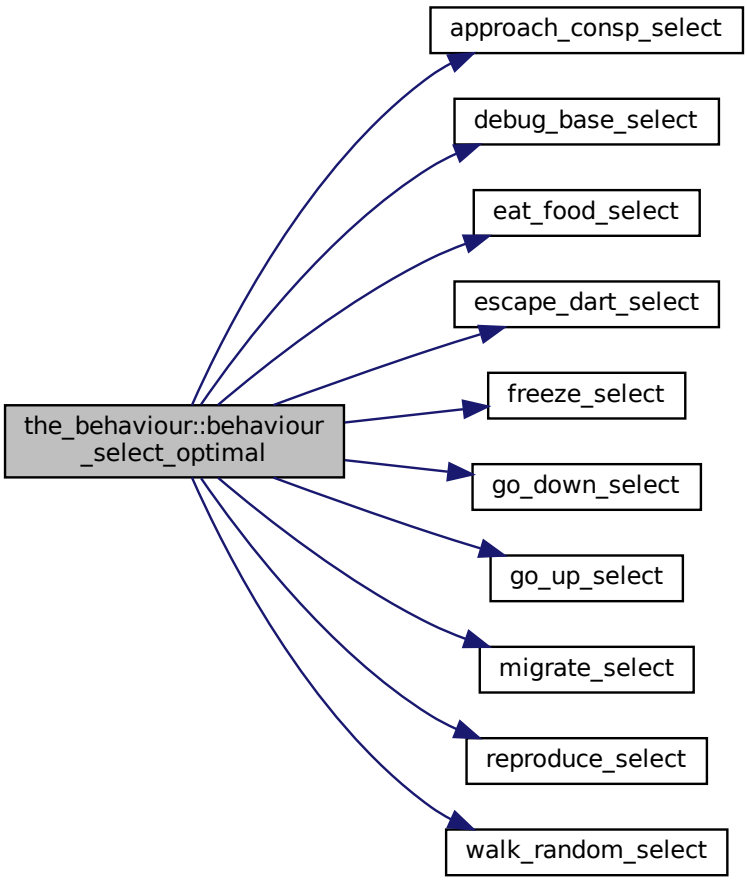
Automatic array allocation is checked. If the ` expected_gos_all array turns out not allocated, a critical error is signalled in the logger.
In the DEBUG mode, the array of the GOS arousal levels is logged.
Second, each of the behaviours is checked for being the minimum value. If true, this behaviour is executed using the `do_` method of the the_behaviour::behaviour class.
Additionally, for each behaviour unit, an additional check is performed to make sure the conditions for the behaviour are satisfied. If the conditions are not satisfied, a default Gausssian random walk the_behaviour::behaviour::do_walk() is done.
The correctness conditions for each of the behaviour units are:

- the_behaviour::eat_food: the agent must have food items in perception, the_neurobio::perception::has_food() is TRUE.

the_behaviour::reproduce: the agent must be mature (the_body::reproduction::is_ready_reproduce() is TRUE) *and* have conspecifics in perception (the_neurobio::perception::has_consp() is TRUE).

- the_behaviour::walk_random: the optimal distance selected `walk_distance_selected` must be nonzero, i.e. exceed the tolerance value commondata::tolerance_high_def_srp.
- the_behaviour::freeze: this behaviour does not require any specific conditions and can be executed anyway.
- the_behaviour::escape_dart: the agent must have predators in its perception, i.e. the_neurobio::perception::has_pred() should return TRUE. If no predators are present, a non-targeted the_behaviour::escape_dart instance is executed.
- the_behaviour::approach_conspec: the agent must have conspecifics in perception, the_neurobio::perception::has_consp() is TRUE.
- the_behaviour::migrate: the agent must have a valid target habitat for migration; the optimal habitat index `habitat_selected_n` must correspond to a valid habitat in the global array the_environment::global_habitats_available.
- the_behaviour::go_down_depth: the optimal vertical migration distance selected `go_down_distance_↩ selected` must be nonzero, i.e. exceed the tolerance value commondata::tolerance_high_def_srp.
- the_behaviour::go_up_depth: the optimal vertical migration distance selected `go_up_distance_↩ selected` must be nonzero, i.e. exceed the tolerance value commondata::tolerance_high_def_srp.

The control is passed back out of this procedure on execution of the optimal behaviour. However, if no behaviour was selected up to this point, the agent just does a default Gaussian walk. However, this situation is very suspicious and can point to a bug. Therefore, such situation is logged with the ERROR tag.
Definition at line 10367 of file m_behav.f90.
Here is the call graph for this function:



8.3.3.67 behaviour_select_fixed_from_gos()

```
subroutine the_behaviour::behaviour_select_fixed_from_gos (
            class(behaviour), intent(inout) this,
            real(srp), intent(in), optional rescale_max_motivation,
            class(food_resource), intent(inout), optional food_resource_real )
```

Select and **execute** behaviour based on the current global organismic state. This procedure is significantly different from the_behaviour::behaviour_select_optimal() in that the behaviour that is executed is not based on optimisation of the expected GOS. Rather, the current GOS fully determines which behaviour unit is executed. Such a rigid link necessarily limits the range of behaviours that could be executed.

Parameters

`in,out`	*food_resource_real*	[inout] food_resource_real The food resource the agent is eating the food item in. Note that it could be a joined food resource composed with the_environment::join() procedure for assembling several habitats into the the_environment::global_habitats_available array or resources collapsed using the the_environment::food_resource::join() method.
`in`	*rescale_max_motivation*	rescale_max_motivation maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

8.3.3.67.1 Notable local variables

- **food_item_selected** is the optimal food item selected from all the items that are currently within the perception object of the agent. In this version of `do_behave`, the nearest food item is selected.

predator_selected_n - the predator object within the perception, that is considered the most subjectively dangerous for the agent. (This is actually the *number* of the predator within the perception object.) Note that in this version of `do_behave`, the nearest predator is selected.

8.3.3.67.2 Implementation details

8.3.3.67.2.1 Checks and preparations Determine optional parameter `rescale_max_motivation`. If it is absent from the parameter list, the value is calculated from the current perception using the the_neurobio::motivation::max_perception() method.

8.3.3.67.2.2 Try to perform random migration Random migration is implemented in the `TRY_MIGRATE` block.

Warning

> This code does not work well in case the agent is within the maximum random migration distance from more than one target environment at once. It cycles in fixed order 1,2... over the commondata::global_habitats_↩ available. Ideally, should select at random. Hopefully, such cases are very rare. TODO.

- First, find what is the current agent's environment within the commondata::global_habitats_available array, calling the_environment::spatial::find_environment() method.
- Second, loop over all the habitats available in the commondata::global_habitats_available array. If the `i`th habitat does not coincide with the current agent's habitat (i.e. the agent cannot emigrate to the currently occupied habitat), the agent tries to perform random migration the_behaviour::behaviour::migrate_random().

Note that the loop is terminated (`exit`) if migration into the i-th habitat was successful. The agent can perform only a single behaviour (migration across habitats) per a single time step.
If the migration was successful, no further behaviour is executed, it is assumed that the agent has executed the_behaviour::migrate behaviour unit.

8.3.3.67.2.3 Execute behaviours depending on the current GOS arousal Fixed behaviour selection is implemented in the `SELECT_BEHAV`construct. Each of the GOS is rigidly associated with a specific behaviour pattern.

- the_neurobio::state_hunger is the GOS:

at least one food item is present within the perception object, calls the the_behaviour::eat_food() method for the nearest food item.

- there are no food items in the perception object, calls default random walk the_behaviour::walk_random.
- the_neurobio::state_fear_defence is the GOS:
- there is at least one predator in perception: calls the_behaviour::escape_dart

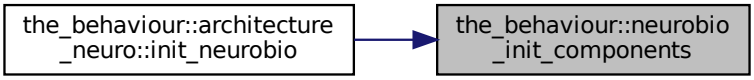
- no predators are present in the perception object: call the_behaviour::freeze.
- the_neurobio::state_reproduce is the GOS:
- if the agent is ready to reproduce and there are conspecifics in proximity, call the_behaviour::reproduce
- if the above condition is not satisfied, do default the_behaviour::walk_random.

Definition at line 11236 of file m_behav.f90.

8.3.3.68 neurobio_init_components()

```
elemental subroutine, private the_behaviour::neurobio_init_components (
            class(architecture_neuro), intent(inout) this )  [private]
```

Initialise neuro-biological architecture.
Initialise neurobiological components of the agent.
Definition at line 11381 of file m_behav.f90.
Here is the caller graph for this function:



8.3.4 Variable Documentation

8.3.4.1 modname

```
character (len=*), parameter, private the_behaviour::modname = "(THE_BEHAVIOUR)"  [private]
```

Definition at line 26 of file m_behav.f90.

8.4 the_body Module Reference

Definition the physical properties and condition of the agent.

Data Types

- type condition

  CONDITION defines the physical condition of the agent
- type reproduction

  REPRODUCTION type defines parameters of the reproduction system.

Functions/Subroutines

- elemental real(srp) function length2mass (k, l)

  This is the function to calculate the body weight from the length and the Fulton condition factor (energy reserves).
- elemental real(srp) function energy_reserve (m, l)

  Calculate the current energy reserves (Fulton condition factor) from body mass and length.
- subroutine condition_init_genotype (this)

*Initialise the individual body condition object based on the genome values. Two alleles are selected at random and input into the `gamma2gene` function to get the initial hormone values rescaled to 0:1. Note that the `gamma2gene` alleles defining the **shape** of the gamma function and the **half-max effect** are selected randomly in this version. Also, polyploid organisms are possible, in such case, two parameters are also randomly defined from a larger set (e.g. from four chromosomes in case of tetraploids). See implementation details and comments for each of the hormones.*

- subroutine birth_mortality_enforce_init_fixed_debug (this)

  This procedure enforces selective mortality of agents at birth to avoid strong selection for energy and length.
- elemental subroutine condition_clean_history (this)

  Cleanup the history stack of the body length and mass.
- elemental integer function condition_age_get (this)

  Get current age. Standard GET-function.
- elemental subroutine condition_age_reset_zero (this)

  Reset the age of the agent to zero.
- elemental subroutine condition_age_increment (this, increment)

  Increment the age of the agent by one.
- elemental real(srp) function condition_energy_current_get (this)

  Get current energy reserves. Standard GET-function.
- elemental real(srp) function condition_energy_maximum_get (this)

  Get historical maximum of energy reserves. Standard GET-function.
- elemental real(srp) function condition_body_length_get (this)

  Get current body length. Standard GET-function.
- elemental real(srp) function condition_control_unsel_get (this)

  Get current value of the control unselected trait. Standard GET-function.
- elemental real(srp) function condition_body_mass_get (this)

  Get current body mass. Standard GET-function.
- real(srp) function condition_agent_visibility_visual_range (this, object_area, contrast, time_step_model)

  Calculate the visibility range of this agent. Visibility depends on the size of the agent, ambient illumination and agent contrast. Visibility is the distance from which this agent can be seen by a visual object (e.g. predator or conspecific). This function is a wrapper to the the_environment::visual_range() function.
- subroutine condition_body_mass_set_update_hist (this, value_set, update_history)

  Set body mass optionally updating the history stack.
- subroutine condition_body_length_set_update_hist (this, value_set, update_history)

  Set body length optionally updating the history stack.
- elemental real(srp) function condition_energy_birth_get (this)

  Get historical record of energy reserves at birth. Standard GET-function.
- elemental real(srp) function condition_body_length_birth_get (this)

  Get historical record of body length at birth. Standard GET-function.
- elemental real(srp) function condition_body_mass_birth_get (this)

  Get historical record of body mass at birth. Standard GET-function.
- elemental real(srp) function condition_body_mass_max_get (this)

  Get historcal maximum for body mass. Standard GET-function.
- elemental real(srp) function condition_smr_get (this)

  Get current smr. Standard GET-function.
- elemental real(srp) function condition_stomach_content_get (this)

  Get current stomach content. Standard GET-function.
- elemental real(srp) real(srp) function body_mass_processing_cost_calc_v (this, food_gain, distance_food)

  Calculate the basic processing cost of catching a food item with the mass `food_gain`.
- elemental real(srp) function condition_cost_swimming_burst (this, distance, exponent)

  The cost of swimming of a specific distance in terms of the actor's body mass.
- elemental real(srp) function body_mass_processing_cost_calc_o (this, food_obj, distance_food)

  Calculate the basic processing cost of catching a food item with the mass `food_gain`.
- elemental real(srp) function stomach_content_food_gain_fitting_v (this, food_gain, food_dist)

Calculate the value of possible food gain as fitting into the agent's stomach, or the full gain if the food item wholly fits in.

- elemental real(srp) function stomach_content_food_gain_fitting_o (this, food_obj, food_dist)

  Calculate the value of possible food gain as fitting into the agent's stomach (or full gain if the food item fits wholly).
- elemental real(srp) function stomach_content_food_gain_non_fit_v (this, food_gain)

  Calculate extra food surplus mass non fitting into the stomach of the agent.
- elemental real(srp) function stomach_content_food_gain_non_fit_o (this, food_obj)

  Calculate extra food surplus mass non fitting into the stomach of the agent.
- elemental real(srp) function body_mass_calculate_cost_living_step (this)

  Calculate the cost of living for a single model step. So the agent mass increment per a single model step should subtract this cost.
- elemental subroutine body_mass_adjust_living_cost_step (this)

  Adjust the body mass at the end of the model step against the cost of living. We do not adjust the cost of living at each food gain as several food items can be consumed by the agent at a single time step of the model. Cost of living is now calculated at the end of the time step of the model.
- elemental subroutine body_mass_grow_do_calculate (this, food_gain, update_history)

  Do grow body mass based on food gain from a single food item adjusted for cost etc.
- elemental real(srp) function body_mass_food_processing_cost_factor_smr (this, food_gain)

  The fraction of the cost of the processing of the food item(s) depending on the agent SMR. It is scaled in terms of the ratio of the food item mass to the agent mass.
- elemental subroutine stomach_content_get_increment (this, stomach_increment)

  Do increment stomach contents with adjusted (fitted) value.
- real(srp) function body_len_grow_calculate_increment_step (this, mass_increment)

  Calculate body length increment for a time step of the model.
- subroutine body_len_grow_do_calculate_step (this, mass_increment, update_history)

  Do linear growth for one model step based on the increment function the_body::condition::len_incr().
- subroutine sex_steroids_update_increment (this)

  Update the level of the sex steroids.
- elemental logical function body_mass_is_starvation_check (this)

  Check if the body mass is smaller than the birth body mass or structural body mass. An agent dies of starvation if either of these conditions is met:
- elemental logical function is_starved (body_mass, stomach_content_mass, body_mass_birth, body_mass↩ _maximum, energy_current, energy_maximum)

  This is the backend logical function that checks if the agent is starved. It is called by the `condition::starved_death()` *=>* `the_body::body_mass_is_starvation_check()` *procedure.*
- elemental subroutine stomach_content_mass_emptify_step (this)

  Digestion. Stomach contents S(t) is emptied by a constant fraction each time step.
- elemental real(srp) function stomach_emptify_backend (stomach_content_mass)

  The backend engine for calculating the stomach content mass decrement as a consequence of digestion. Stomach contents S(t) is emptied by a constant fraction each time step ΔS*:*
- elemental subroutine condition_energy_update_after_growth (this)

  Update the energy reserves of the agent based on its current mass and length. This subroutine should be called after any event that can change the mass or/and length of the agent, e.g. food consumption.
- elemental real(srp) function cost_swimming_standard (this, steps)

  The standard cost of swimming is a diagnostic function that shows the cost, in units of the body mass, incurred if the agent passes a distance equal to commondata::lifespan units of its body length.
- elemental real(srp) function reproduction_cost_energy_fix (this)

  Calculate the energetic cost of reproduction in terms of the ***body mass*** *of the this agent. The energetic cost of reproduction is obtained as a specific fixed fraction of the current body mass of the agent defined by the* `commondata↩ ::reproduction_cost_body_mass` *parameter.*
- real(srp) function reproduction_cost_energy_dynamic (this)

  Calculate the energetic cost of reproduction in terms of the ***body mass*** *of the this agent. The energetic cost of reproduction is different in males and females.*

- real(srp) function reproduction_cost_unsuccessful_calc (this, cost_factor)

  Calculate the costs of unsuccessful reproduction. This is calculated as a fraction of the normal cost of reproduction returned by the function `reproduction::reproduction_cost()`. Reproduction can be unsuccessful for various reasons: insufficient reserves (reproduction results in starvation death) or stochastic no success.
- elemental subroutine reproduction_init_zero (this)

  Initialise the reproduction object for the agent. Everything is set to zero.
- elemental integer function reproduction_n_reproductions_get (this)

  Get the number of reproductions for this agent.
- elemental subroutine reproduction_n_reproductions_set (this, n_repr)

  Set the number of reproductions for the agent.
- elemental integer function reproduction_n_offspring_get (this)

  Get the number of offspring.
- elemental subroutine reproduction_n_offspring_set (this, n_offspr)

  Set the number of offspring for the agent.
- subroutine reproduction_n_increment (this, add_repr, average_mass_offspring)

  Increment the number of reproductions and offspring for this agent.
- integer function reproduction_n_offspring_calc (this, average_mass_offspr)

  Calculate the number of offspring per a single reproduction as a function of the agent's body mass.
- elemental logical function reproduction_ready_steroid_hormones_exceed (this)

  Determine if the agent's hormonal system is ready for reproduction.
- real(srp) function reproduction_mass_offspring_calc (this)

  Calculate the total mass of all offspring produced by this agent during a single reproduction event.

Variables

- character(len= ∗), parameter, private modname = "(THE_CONDITION)"

8.4.1 Detailed Description

Definition the physical properties and condition of the agent.

8.4.2 THE_BODY module

This module defines various physical properties of the agent, such as the body size, body mass etc, as well as the condition and basic physiological variables.

Note

Note that the agent has the size property but is nonetheless represented as a single commondata::spatial point for simplicity.

8.4.3 Function/Subroutine Documentation

8.4.3.1 length2mass()

```
elemental real(srp) function the_body::length2mass (
            real(srp), intent(in) k,
            real(srp), intent(in) l )
```

This is the function to calculate the body weight from the length and the Fulton condition factor (energy reserves).

Parameters

in	*k,l*	condition and body length.

Returns

Body mass.

8.4.3.1.1 Implementation details Body mass is non-genetic, length and initial condition factor are genetically determined. Body mass is calculated initially from the Fulton'scondition factor formula

$$K = \frac{M}{L^3},$$

i.e.

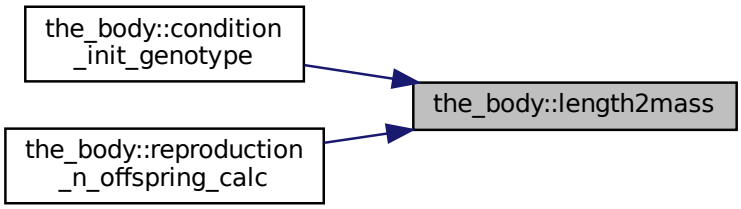
$$M = KL^3.$$

The exponent can be non-cube for non-isometric growth.

Note

The "cube law" exponent (3.0 normally), might be redefined here as the LINEAR_GROWTH_EXPONENT parameter constant.

Definition at line 308 of file m_body.f90.
Here is the caller graph for this function:



8.4.3.2 energy_reserve()

```
elemental real(srp) function the_body::energy_reserve (
            real(srp), intent(in) m,
            real(srp), intent(in) l )
```

Calculate the current energy reserves (Fulton condition factor) from body mass and length.

Parameters

in	*m,l*	body mass and body length.

Returns

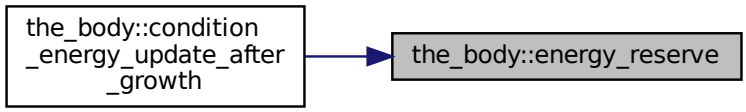
energy reserve available.

Note

The "cube law" exponent (3.0 normally), might be redefined here as the LINEAR_GROWTH_EXPONENT parameter constant.

Definition at line 331 of file m_body.f90.

Here is the caller graph for this function:



8.4.3.3 condition_init_genotype()

```
subroutine the_body::condition_init_genotype (
            class(condition), intent(inout) this )
```

Initialise the individual body condition object based on the genome values. Two alleles are selected at random and input into the `gamma2gene` function to get the initial hormone values rescaled to 0:1. Note that the `gamma2gene` alleles defining the **shape** of the gamma function and the **half-max effect** are selected randomly in this version. Also, polyploid organisms are possible, in such case, two parameters are also randomly defined from a larger set (e.g. from four chromosomes in case of tetraploids). See implementation details and comments for each of the hormones.

8.4.3.3.1 Implementation details First, initialise all the physical condition components of the agent, starting from **age**: age=0 initially.
The **energy reserves** are set as Gaussian with the mean commondata::energy_init and CV commondata::energy_gerror_cv.
Set the birth energy reserves from the initial current value.
Additionally, update the historical maximum energy value.
The **body length** is initialised as Gaussian with the mean commondata::body_length_init and cv commondata::body_length_gerror_cv

Note

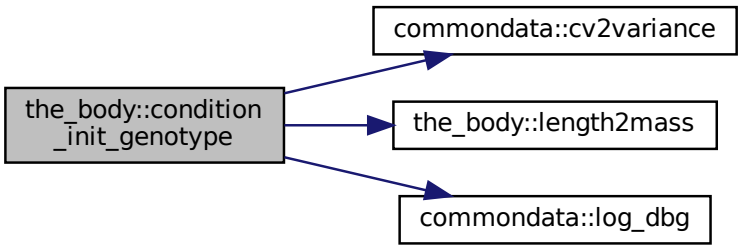
Body length cannot be zero or less than the minimum possible size that is defined by `BODY_LENGTH_MIN`.

Also, body length at birth cannot reach the maximum value `BODY_LENGTH_MAX`, if it does occurs, erroneous parameter value was set. This aberrant agent then the_genome::individual_genome::dies().
The historical body length at birth is saved as the_body::condition::body_length_birth.
A **control unselected** trait is also set from the genome. This trait is not used in any calculations but serves as a control for random or nonrandom genetic drift.
The **body mass** is determined by the genetically determined energy reserves and the body length (using `length2mass` function). Thus, the body mass is **non-genetic**.
The historical body mass at birth and the maximum body mass ever achieved are saved.
SMR is set from the genome.
However, it must never be lower than commondata::smr_min. Very low values are unrealistic and might crash model.
Stomach contents is initialised as a random Gaussian value, average, units of the body mass with `STOMACH_CONTENT_INIT` and coefficient of variation `STOMACH_CONTENT_INIT_CV`. Stomach contents also must always be above zero and never exceed the `maxstomcap` factor.
Finally, the procedure initialises the history stacks for the body mass and length.
And put the initial birth values of body length and mass into the history stack.

Note

The body length and mass history stack keeps the latest historical values.

Definition at line 354 of file m_body.f90.
Here is the call graph for this function:



8.4.3.4 birth_mortality_enforce_init_fixed_debug()

```
subroutine the_body::birth_mortality_enforce_init_fixed_debug (
             class(condition), intent(inout) this )
```

This procedure enforces selective mortality of agents at birth to avoid strong selection for energy and length.

Warning

This is a debug version of the mortality procedure with fixed mortality pattern, final should depend on the statistical properties of the first generation, mean and sd.

Definition at line 463 of file m_body.f90.

8.4.3.5 condition_clean_history()

```
elemental subroutine the_body::condition_clean_history (
             class(condition), intent(inout) this )
```

Cleanup the history stack of the body length and mass.
Definition at line 489 of file m_body.f90.

8.4.3.6 condition_age_get()

```
elemental integer function the_body::condition_age_get (
             class(condition), intent(in) this )
```

Get current age. *Standard GET-function.*

Returns

Return the agent's age

Definition at line 502 of file m_body.f90.

8.4.3.7 condition_age_reset_zero()

```
elemental subroutine the_body::condition_age_reset_zero (
            class(condition), intent(inout) this )
```

Reset the age of the agent to zero.
Definition at line 513 of file m_body.f90.

8.4.3.8 condition_age_increment()

```
elemental subroutine the_body::condition_age_increment (
            class(condition), intent(inout) this,
            integer, intent(in), optional increment )
```

Increment the age of the agent by one.

Parameters

in	*increment*	increment optional increment for increasing the age of the agent, the default value is 1.

Definition at line 522 of file m_body.f90.

8.4.3.9 condition_energy_current_get()

```
elemental real(srp) function the_body::condition_energy_current_get (
            class(condition), intent(in) this )
```

Get current energy reserves. *Standard GET-function.*

Returns

Return the agent's energy reserves.

Definition at line 538 of file m_body.f90.

8.4.3.10 condition_energy_maximum_get()

```
elemental real(srp) function the_body::condition_energy_maximum_get (
            class(condition), intent(in) this )
```

Get historical maximum of energy reserves. *Standard GET-function.*

Returns

Return the agent's maximum energy reserves.

Definition at line 549 of file m_body.f90.

8.4.3.11 condition_body_length_get()

```
elemental real(srp) function the_body::condition_body_length_get (
            class(condition), intent(in) this )
```

Get current body length. *Standard GET-function.*

Returns

Return the agent's body length.

Definition at line 560 of file m_body.f90.

8.4.3.12 condition_control_unsel_get()

```
elemental real(srp) function the_body::condition_control_unsel_get (
             class(condition), intent(in) this )
```

Get current value of the control unselected trait. Standard GET-function.

Returns

Return the agent's control unselected trait value.

Definition at line 571 of file m_body.f90.

8.4.3.13 condition_body_mass_get()

```
elemental real(srp) function the_body::condition_body_mass_get (
             class(condition), intent(in) this )
```

Get current body mass. *Standard GET-function.*

Returns

Return the agent's body mass.

Definition at line 582 of file m_body.f90.

8.4.3.14 condition_agent_visibility_visual_range()

```
real(srp) function the_body::condition_agent_visibility_visual_range (
             class(condition), intent(in) this,
             real(srp), intent(in), optional object_area,
             real(srp), intent(in), optional contrast,
             integer, intent(in), optional time_step_model )
```

Calculate the visibility range of this agent. Visibility depends on the size of the agent, ambient illumination and agent contrast. Visibility is the distance from which this agent can be seen by a visual object (e.g. predator or conspecific). This function is a wrapper to the the_environment::visual_range() function.

Warning

The `visual_range` procedures use meter for units, this auto-converts to cm.

Cannot implement a generic function accepting also vectors of this objects as only elemental object-bound array functions are allowed by the standard. This function cannot be elemental, so passed-object dummy argument must always be scalar.

Parameters

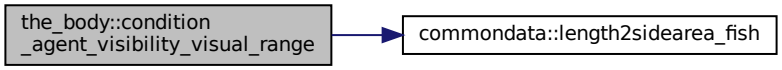
in	*object_area*	object_area optional area of this agent, m. If not provided (normally), is obtained from the body length attribute of the agent (the_body::condition::body_length).
in	*contrast*	contrast is the inherent visual contrast of the agent. the default contrast of all objects is defined by the commondata::preycontrast_default parameter.
in	*time_step_model*	optional time step of the model, if absent gets the current time step as defined by the value of `commondata::global_time_step_model_current`.

Returns

The maximum distance from which this agent can be seen.

8.4.3.14.1 Implementation details **Checks.** Check optional object area, the default value, if this parameter is absent, the body side area is calculated from the the_body::condition::body_length attribute of the agent with inline conversion to m. Note that the body side area of a fish object is calculated from the body length using the commondata::length2sidearea_fish() function.

Check optional contrast parameter. If unset, use global commondata::preycontrast_default.
Check optional time step parameter. If unset, use global commondata::global_time_step_model_current.
Calculate ambient illumination / irradiance at the depth of this agent at the given time step using the the_environment::spatial::illumination() method.
Return visual range to see this spatial object: its visibility range by calling the the_environment::visual_range() function.
Definition at line 603 of file m_body.f90.
Here is the call graph for this function:



8.4.3.15 condition_body_mass_set_update_hist()

```
subroutine the_body::condition_body_mass_set_update_hist (
            class(condition), intent(inout) this,
            real(srp), intent(in) value_set,
            logical, intent(in), optional update_history )
```

Set body mass optionally updating the history stack.

Parameters

in	*value_set*	value_set, Set the new (overwrite) value of the **body mass**.
in	*update_history*	update_history is an optional logical flag to update the body mass history stack, the default is **not to update**.

8.4.3.15.1 Implementation details If the value_set is smaller that the minimum body mass parameter BODY_MASS_MIN, the body mass is set to this minimum value. This avoids getting the body mass too small or negative. This "set"-procedure, however, does not check if the new value is below the structure mass or any other minimum value that leads to the death of the agent. To check for starvation death, the method condition::starved_death() => the_body::body_mass_is_starvation_check() should be explicitly executed.
Update the body mass history stack if the update_history is explicitly set to TRUE. The default not to update is used because body mass should normally be updated in parallel with the length, if this is not the case, they will be dis-synchronised within the history stack arrays.
Definition at line 672 of file m_body.f90.

8.4.3.16 condition_body_length_set_update_hist()

```
subroutine the_body::condition_body_length_set_update_hist (
            class(condition), intent(inout) this,
            real(srp), intent(in) value_set,
            logical, intent(in), optional update_history )
```

Set body length optionally updating the history stack.

Parameters

in	*value_set*	value_set, Set the new (overwrite) value of the **body length**.

Parameters

in	*update_history*	update_history is an optional logical flag to update the body length history stack, the default is **not to update**.

8.4.3.16.1 Implementation details If the `value_set` is smaller that the minimum body length parameter `BODY_LENGTH_MIN` or the maximum `BODY_LENGTH_MAX`, the length is set to this minimum or maximum value respectively. This avoids setting the body length outside of the normal limits. The function `commondata::within()` is called to set the new value.
Update the body length history stack if the `update_history` is explicitly set to TRUE. The default not to update is used because body length should normally be updated in parallel with the mass, if this is not the case, they will be dis-synchronised within the history stack arrays.
Definition at line 711 of file m_body.f90.

8.4.3.17 condition_energy_birth_get()

```
elemental real(srp) function the_body::condition_energy_birth_get (
             class(condition), intent(in) this )
```

Get historical record of energy reserves at birth. *Standard GET-function.*

Returns

Return the agent's body length at birth.

Definition at line 742 of file m_body.f90.

8.4.3.18 condition_body_length_birth_get()

```
elemental real(srp) function the_body::condition_body_length_birth_get (
             class(condition), intent(in) this )
```

Get historical record of body length at birth. *Standard GET-function.*

Returns

Return the agent's body length at birth.

Definition at line 753 of file m_body.f90.

8.4.3.19 condition_body_mass_birth_get()

```
elemental real(srp) function the_body::condition_body_mass_birth_get (
             class(condition), intent(in) this )
```

Get historical record of body mass at birth. *Standard GET-function.*

Returns

Return the agent's body mass at birth.

Definition at line 764 of file m_body.f90.

8.4.3.20 condition_body_mass_max_get()

```
elemental real(srp) function the_body::condition_body_mass_max_get (
             class(condition), intent(in) this )
```

Get historcal maximum for body mass. Standard *GET-function.*

Returns

Return the agent's maximum body mass.

Definition at line 775 of file m_body.f90.

8.4.3.21 condition_smr_get()

```
elemental real(srp) function the_body::condition_smr_get (
            class(condition), intent(in) this )
```

Get current smr. Standard *GET-function.*

Returns

Return the agent's SMR.

Definition at line 786 of file m_body.f90.

8.4.3.22 condition_stomach_content_get()

```
elemental real(srp) function the_body::condition_stomach_content_get (
            class(condition), intent(in) this )
```

Get current stomach content. *Standard GET-function.*

Returns

Return the agent's stomach content.

Definition at line 797 of file m_body.f90.

8.4.3.23 body_mass_processing_cost_calc_v()

```
elemental real(srp) real(srp) function the_body::body_mass_processing_cost_calc_v (
            class(condition), intent(in) this,
            real(srp), intent(in), optional food_gain,
            real(srp), intent(in), optional distance_food )
```

Calculate the basic processing cost of catching a food item with the mass `food_gain`.
There is a small cost of the food item catching, in terms of the **food item mass** (proportional cost). So, if the agent does an unsuccessful attempt to catch a food item, the cost still applies. So we subtract it before testing if the agent actually got this food item. Also, there is a fixed minimum capture cost (in terms of the **agent body mass**), so if the food item is very small, the actual gain can be negative (capture cost exceeds the value of the item).

Note

Note that this version accepts the the raw food mass (real value).

Parameters

in	*distance_food*	distance_food distance to the food item.

8.4.3.23.1 Implementation details First, check the optional distance towards the food item. It is used to calculate the energetic cost of swimming towards the food item.
If the distance to the food item is not provided, we assume it is equal to the *agent size* (so the relative distance = 1 body size).
The cost of the processing of the food item is a sum of two components:

1. some small processing cost depending on the food item mass and

2. the cost of swimming towards the food item depending on the relative distance (distance in terms of the agent body length.

$$C_p = max(\mu \cdot \beta_{fp}, \mu \cdot C_{smr}) + C_s,$$

where μ is the food gain, β_{fp} is a factor proportional to the food item mass, and C_{smr} is a food processing cost factor that is proportional to the agent's SMR.
Definition at line 819 of file m_body.f90.

8.4.3.24 condition_cost_swimming_burst()

```
elemental real(srp) function the_body::condition_cost_swimming_burst (
             class(condition), intent(in) this,
             real(srp), intent(in), optional distance,
             real(srp), intent(in), optional exponent )
```

The cost of swimming of a specific distance in terms of the actor's body mass.

Note

Note that power needed to swim is proportional to the body mass with the exponent 0.6 assuming turbulent flow (see doi:10.1242/jeb.01484).

Parameters

in	*distance*	the optional distance traversed (absolute distance in real units, cm). If distance is not provided, it is calculated from the latest spatial displacement of the agent using the the_environment::spatial_moving::way() function.
in	*exponent*	an optional cost exponent parameter. Can be 0.5 (commondata::swimming_cost_exponent_laminar, laminar flow) or 0.6 (commondata::swimming_cost_exponent_turbulent, turbulent flow), the default is set to 0.6.

Returns

The cost of swimming in terms of the body mass lost.

8.4.3.24.1 Notable parameters **SWIM_COST_EXP** is the default swimming cost body mass exponent parameter for turbulent flow commondata::swimming_cost_exponent_turbulent = 0.6. For laminar flow, equal to commondata::swimming_cost_exponent_laminar = 0.5. See doi:10.1242/jeb.01484 (https://dx.doi.org/10.1242/jeb.01484).

8.4.3.24.2 Implementation details The cost of swimming (for turbulent flow) is calculated as:

$$C_s = M^{0.6} \cdot \beta \cdot d/L,$$

where M is the body mass, β is a parameter factor defined as commondata::swimming_speed_cost_burst, d/L is the distance in units of the agent's body length. For laminar flow, the exponent should be 0.5.

Note

An arbitrary value for the exponent can be provided as the second dummy parameter to this function exponent.

The function the_body::cost_swimming_standard() calculates a diagnostic function, the "standard" cost of swimming.

Definition at line 879 of file m_body.f90.

8.4.3.25 body_mass_processing_cost_calc_o()

```
elemental real(srp) function the_body::body_mass_processing_cost_calc_o (
            class(condition), intent(in) this,
            class(food_item), intent(in) food_obj,
            real(srp), intent(in), optional distance_food )
```

Calculate the basic processing cost of catching a food item with the mass `food_gain`.

Note

Note that this version accepts the food object not its raw mass.

Parameters

in	*food_obj*	food item object, of class `FOOD_ITEM`.
in	*distance_food*	distance to the food item.

Returns

Food processing cost.

8.4.3.25.1 Implementation details First, check the optional distance towards the food item. We use it to calculate the energetic cost of swimming towards the food item.
If the distance to the food item is not provided, we assume it is equal to the agent body size (so the relative distance = 1 body size).
The cost of the processing of the food item is a sum of two components:

1. some small processing cost depending on the food item mass and
2. the cost of swimming towards the food item depending on the relative distance (distance in terms of the agent body length.

$$C_p = max(\mu \cdot \beta_{fp}, \mu \cdot C_{smr}) + C_s,$$

where μ is the food gain, β_{fp} is a factor proportional to the food item mass, and C_{smr} is a food processing cost factor that is proportional to the agent's SMR.

Note

The calculations are done by the scalar procedure body_mass_processing_cost_calc_v().

Definition at line 934 of file m_body.f90.

8.4.3.26 stomach_content_food_gain_fitting_v()

```
elemental real(srp) function the_body::stomach_content_food_gain_fitting_v (
            class(condition), intent(in) this,
            real(srp), intent(in), optional food_gain,
            real(srp), intent(in), optional food_dist )
```

Calculate the value of possible food gain as fitting into the agent's stomach, or the full gain if the food item wholly fits in.

Parameters

in	*food_gain*	food gain.
in	*food_dist*	distance to food.

Returns

processing cost.

Note

Note that this version accepts the the raw food mass (real value).

The food fitting is adjusted for the food item processing cost body_mass_processing_cost_calc_v() call.

8.4.3.26.1 Implementation details Check optional `food_gain` parameter, set default values. If food gain is not provided, an average/default food item is assumed, defined by `FOOD_ITEM_SIZE_DEFAULT`.
Definition at line 988 of file m_body.f90.

8.4.3.27 stomach_content_food_gain_fitting_o()

```
elemental real(srp) function the_body::stomach_content_food_gain_fitting_o (
             class(condition), intent(in) this,
             class(food_item), intent(in) food_obj,
             real(srp), intent(in), optional food_dist )
```

Calculate the value of possible food gain as fitting into the agent's stomach (or full gain if the food item fits wholly).

Note

Note that this version accepts the food object not its raw mass.

Note that the food fitting is adjusted for the food item processing cost via `food_process_cost` call.

This code is **not using** the `food_fitt_v` scalar-based function above: `if (present(food_dist)) then food_↩ adjusted = food_objget_mass() - thisfood_surplus(food_obj) - & thisfood_process_cost(food_obj, food_dist) else food_adjusted = food_objget_mass() - thisfood_surplus(food_obj) - & thisfood_process_cost(food_obj) end if`
This code **uses** the `food_fitt_v` scalar-based function above to avoid code duplication.
Definition at line 1024 of file m_body.f90.

8.4.3.28 stomach_content_food_gain_non_fit_v()

```
elemental real(srp) function the_body::stomach_content_food_gain_non_fit_v (
             class(condition), intent(in) this,
             real(srp), intent(in), optional food_gain )
```

Calculate extra food surplus mass non fitting into the stomach of the agent.

Note

Note that this version accepts the the raw food mass (real value).

Maximum stomach capacity is determined by the factor `maxstomcap` in proportion of the body mass. The stomach content cannot surpass `maxstomcap`=15% of agent's body mass.
Check optional parameter, set default values.
Get the possible food surplus, the part of the food gain that does not fit into the agent's stomach. If happily fits, takes zero.
Definition at line 1056 of file m_body.f90.

8.4.3.29 stomach_content_food_gain_non_fit_o()

```
elemental real(srp) function the_body::stomach_content_food_gain_non_fit_o (
             class(condition), intent(in) this,
             class(food_item), intent(in) food_obj )
```

Calculate extra food surplus mass non fitting into the stomach of the agent.

Note

Note that this version accepts the food object not its raw mass.

Maximum stomach capacity is determined by the factor `maxstomcap` in proportion of the body mass. The stomach content cannot surpass `maxstomcap`=15% of agent's body mass.
Get the possible food surplus, the part of the food gain that does not fit into the agent's stomach. If happily fits, takes zero.
Definition at line 1089 of file m_body.f90.

8.4.3.30 body_mass_calculate_cost_living_step()

```
elemental real(srp) function the_body::body_mass_calculate_cost_living_step (
            class(condition), intent(in) this )
```

Calculate the cost of living for a single model step. So the agent mass increment per a single model step should subtract this cost.

Note

Should be calculated at the end of model time step.

The energetic costs is a fraction of body weight and scales to number of time steps :

$$M_c = \frac{CB}{\Omega}$$

(eq. 1)
Definition at line 1126 of file m_body.f90.

8.4.3.31 body_mass_adjust_living_cost_step()

```
elemental subroutine the_body::body_mass_adjust_living_cost_step (
            class(condition), intent(inout) this )
```

Adjust the body mass at the end of the model step against the cost of living. We do not adjust the cost of living at each food gain as several food items can be consumed by the agent at a single time step of the model. Cost of living is now calculated at the end of the time step of the model.

Note

Should be calculated at the end of model time step.

Definition at line 1149 of file m_body.f90.

8.4.3.32 body_mass_grow_do_calculate()

```
elemental subroutine the_body::body_mass_grow_do_calculate (
            class(condition), intent(inout) this,
            real(srp), intent(in) food_gain,
            logical, intent(in), optional update_history )
```

Do grow body mass based on food gain from a single food item adjusted for cost etc.

Note

Can be calculated after consumption of each food item, many times within a single time step of the model.

Parameters

in	*update_history*	update_history optional logical flag to enable saving the body mass value to the body mass history stack.

Warning

History update is disabled by default because the length and mass histories can be updated separately, so could get not synchronous.

Add the food increment to the current body mass.
And also update the historical maximum value, if the current exceeds.
Add the current updated body mass to the history stack.
Definition at line 1161 of file m_body.f90.

8.4.3.33 body_mass_food_processing_cost_factor_smr()

```
elemental real(srp) function the_body::body_mass_food_processing_cost_factor_smr (
            class(condition), intent(in) this,
            real(srp), intent(in) food_gain )
```

The fraction of the cost of the processing of the food item(s) depending on the agent SMR. It is scaled in terms of the ratio of the food item mass to the agent mass.
Definition at line 1190 of file m_body.f90.

8.4.3.34 stomach_content_get_increment()

```
elemental subroutine the_body::stomach_content_get_increment (
            class(condition), intent(inout) this,
            real(srp), intent(in) stomach_increment )
```

Do increment stomach contents with adjusted (fitted) value.

Note

Note that the `stomach_increment` should be adjusted for fitting size. This is the actual increment.

8.4.3.34.1 Implementation details Stomach content mass is incremented by the `stomach_increment` value.
Definition at line 1205 of file m_body.f90.

8.4.3.35 body_len_grow_calculate_increment_step()

```
real(srp) function the_body::body_len_grow_calculate_increment_step (
            class(condition), intent(in) this,
            real(srp), intent(in) mass_increment )
```

Calculate body length increment for a time step of the model.
This function describes linear growth of the agent resulting from food intake. Linear growth increment scales with **growth hormone** level. The increment in length is weighted by the growth hormone factor that is obtained via interpolation. So, if the agent's growth hormone level is very low, the growth increment factor is near-zero, if growth hormone level is high, the growth increment approaches the maximum value that is proportional to the body mass increment in units of the agent's body mass (growth hormone weighting factor approaches 1.0).

8.4.3.35.1 Notable local parameters `increment_factor_ipoint` is a local parameter showing the linear growth increment factor interpolation point, ϑ in the formula below.

8.4.3.35.2 Implementation details If the body mass increment is positive and exceeds fixed threshold `MASS_GROWTH_THRESHOLD`, the agent can grow body length. Otherwise, if the mass threshold is not exceeded in MASS_THRESHOLD, the agent does not increase in length. This check is done in the main named if condition block `MASS_THRESHOLD`.

8.4.3.35.2.1 MASS_THRESHOLD block

- **First,** we get the interpolation-based growth increment factor `increment_factor_ipoint` (ϑ) depending on the agent's current **growth hormone** level. The function linking growth hormone level and the linear growth increment factor (ϑ) is represented by this relationship:

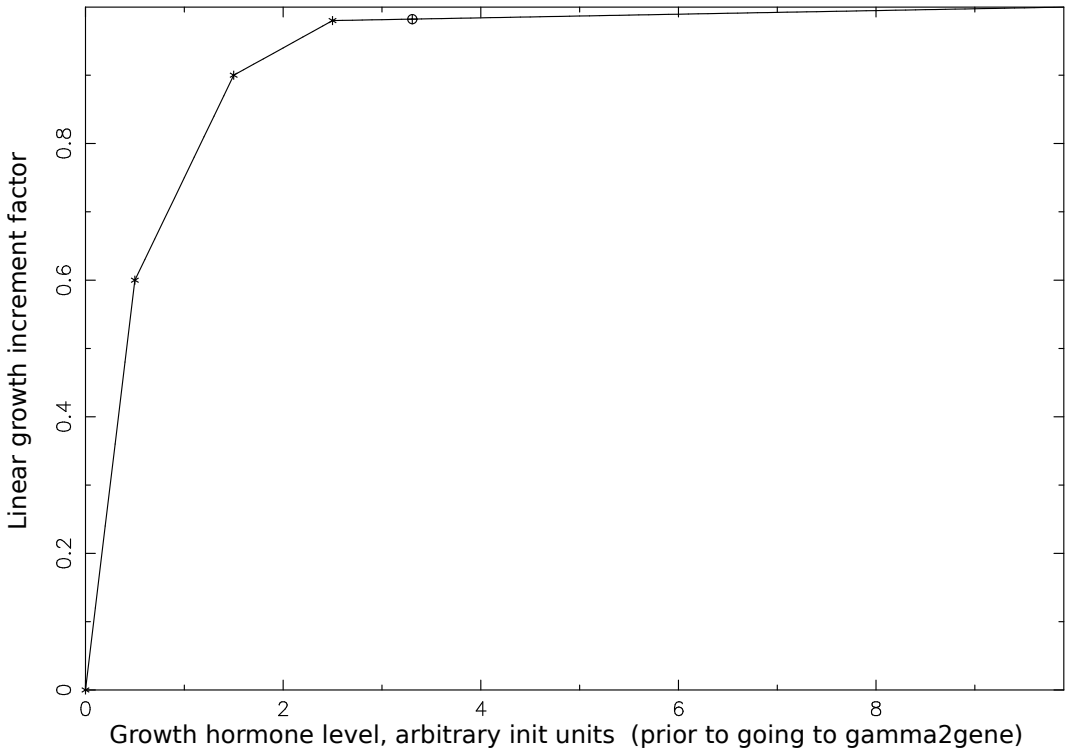


Figure 8.5 Linear growth increment factor

Note

Note that the linear interpolation `LINTERPOL` engine is used here instead of non-linear `DDPINTERPOL`. This is done because of not well predictable raw values in the grid abscissa; `DDPINTERPOL` tends to produce sigmoidal waves here and needs precise tuning of the interpolation grid parameter arrays

- `LINEAR_GROWTH_HORMONE_INCREMENT_FACTOR_CURVE_ABSCISSA`
- `LINEAR_GROWTH_HORMONE_INCREMENT_FACTOR_CURVE_ORDINATE`.

- Save the interpolation plot in the debug mode using external command.

Warning

Involves **huge** number of plots, should normally be disabled.

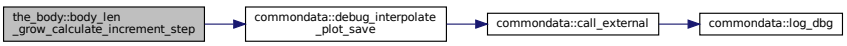
- **Second,** The body length increment in units of length $\frac{\Delta L}{L}$ is proportional to the body mass increment in units of mass: $\frac{\Delta M}{M}$, however weighted by the linear growth increment factor ϑ that depends on the growth hormone and is obtained via interpolation (see above). If the agent's growth hormone level is very low, the growth increment factor is near-zero and linear growth increment is also near-zero. However, if growth hormone level is high, the growth increment weighting factor ϑ approaches the maximum value that is proportional to the body mass increment in units of the agent's body mass:

$$\Delta L = L \cdot \vartheta \cdot \frac{\Delta M}{M},$$

where ΔL is the body length increment, L is the body length, ϑ is the growth-hormone-dependent linear growth increment factor, $\frac{\Delta M}{M}$ is the body mass increment in units of body mass. So the relative body length increment is proportional to the relative body mass increment.

- If the mass threshold is not exceeded in MASS_THRESHOLD, the agent does not increase in length, the length increment is zero.

Definition at line 1229 of file m_body.f90.
Here is the call graph for this function:



8.4.3.36 body_len_grow_do_calculate_step()

```
subroutine the_body::body_len_grow_do_calculate_step (
             class(condition), intent(inout) this,
             real(srp), intent(in) mass_increment,
             logical, intent(in), optional update_history )
```

Do linear growth for one model step based on the increment function the_body::condition::len_incr().

Note

Should be calculated at the end of model time step.

Parameters

in	*update_history*	update_history optional logical flag to enable saving the body mass value to the body mass history stack.

Warning

History update is disabled by default because the length and mass histories can be updated separately, so could get not synchronous.

8.4.3.36.1 Implementation details Body length is incremented by a value of the the_body::condition::len_incr() function.
Also, the current updated body length is added to the history stack.
Definition at line 1320 of file m_body.f90.

8.4.3.37 sex_steroids_update_increment()

```
subroutine the_body::sex_steroids_update_increment (
             class(condition), intent(inout) this )
```

Update the level of the sex steroids.
Sex steroids are incremented each model time step. Testosteron is incremented in males and estrogen, in females. However, such an increment occurs only if there has recently been any body length growth (which occurs only if the food gain exceeds a specific threshold value). The growth increment is calculated as the difference between the current body length and the maximum body length in `n` latest historical entries from the length history stack. The `n` value is set by the parameter commondata::sex_steroids_check_history.

8.4.3.37.1 Implementation details First, the sex steroid **increment factor** is calculated using a nonparametric relationship. Calculations can be based either on its link with the **age** or the **body length**:

- steroid_factor_age() or
- steroid_factor_len().

These are the two alternative procedures implemented here.

Note

Here implementation is based on steroid_factor_age().

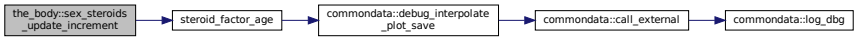
Next, calculate the **past increments of the body length** across the body length history stack. If there has been any increment in the body length during the commondata::sex_steroids_check_history latest steps in the history stack **and** the current length, sex steroids are incremented. If such an increment is zero, sex steroids are not incremented. The length increment over the latest history is calculated as follows:

```
length_increment =                                          &
  maxval( [history_array, current_value] ) -                &
  minval( [history_array, current_value] )
```

Finally, do increment the sex steroids depending on the body length (`length_increment`) value. Sex steroids get non-zero increment only if there has been some growth of the body length (`length_increment>0.0`). Otherwise a previous value is retained.

- If the agent is **male**, **testosterone** is incremented.
- If the agent is **female**, **estrogen** is incremented.

If there was no growth and the gonadal steroids are not incremented, the current values are still saved in the history stack by calling the_hormones::hormones::hormones_to_history().
Definition at line 1354 of file m_body.f90.
Here is the call graph for this function:



8.4.3.38 body_mass_is_starvation_check()

```
elemental logical function the_body::body_mass_is_starvation_check (
             class(condition), intent(in) this )
```

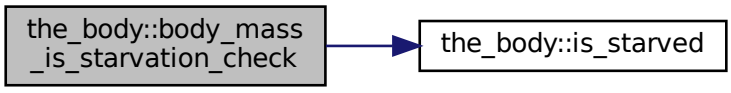
Check if the body mass is smaller than the birth body mass or structural body mass. An agent dies of starvation if either of these conditions is met:

- its body mass falls below half the birth mass;
- below the structural mass, which is defined as half the historic maximum body mass of the individual;
- energy reserves fall below 1/4 of historical maximum value;
- body mass is below the hard limit BODY_MASS_MIN.

Returns

Returns a logical flag: TRUE if starved, FALSE otherwise.

The `the_body::is_starved()` backend function (non-OO) is called to check the starvation condition.
Definition at line 1538 of file m_body.f90.
Here is the call graph for this function:



8.4.3.39 is_starved()

```
elemental logical function the_body::is_starved (
            real(srp), intent(in) body_mass,
            real(srp), intent(in) stomach_content_mass,
            real(srp), intent(in) body_mass_birth,
            real(srp), intent(in) body_mass_maximum,
            real(srp), intent(in) energy_current,
            real(srp), intent(in) energy_maximum )
```

This is the backend logical function that checks if the agent is starved. It is called by the `condition::starved_death()` => `the_body::body_mass_is_starvation_check()` procedure.

Note

Note that this function is not type-bound (non-OO).

Parameters

in	*body_mass*	body_mass the current body mass of the agent.
in	*stomach_content_mass*	stomach_content_mass the mass of the stomach content of the agent.
in	*body_mass_birth*	body_mass_birth body mass of the agent at birth.
in	*body_mass_maximum*	body_mass_maximum the historical maximum body mass of the agent.
in	*energy_current*	energy_current the current level of energy.
in	*energy_maximum*	energy_maximum the historical maximum level of energy.

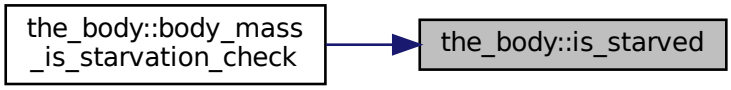
Returns

TRUE if the input conditions make the agent starved to death. Otherwise returns FALSE.

8.4.3.39.1 Conditions of severe starvation An agent is considered starving to death if either of these conditions is met:

- its body mass falls below half the birth mass;
- below the structural mass, which is defined as half the historic maximum body mass of the individual;
- energy reserves fall below 1/4 of historical maximum value;
- body mass is below the hard limit BODY_MASS_MIN.

Definition at line 1561 of file m_body.f90.
Here is the caller graph for this function:



8.4.3.40 stomach_content_mass_emptify_step()

```
elemental subroutine the_body::stomach_content_mass_emptify_step (
            class(condition), intent(inout) this )
```

Digestion. Stomach contents S(t) is emptied by a constant fraction each time step.

$$S_{t+1} = S_t - S_t \frac{K}{\Omega},$$

where K is the *stomach content emptify factor* parameter (commondata::stomach_content_emptify_factor) and Ω is the lifespan (commondata::lifespan parameter). The calculation calls the backend function for $\Delta S = S_t \frac{K}{\Omega}$: the_body::stomach_emptify_backend().

Note

Should be calculated at the end of model time step.

Definition at line 1615 of file m_body.f90.
Here is the call graph for this function:

8.4.3.41 stomach_emptify_backend()

```
elemental real(srp) function the_body::stomach_emptify_backend (
            real(srp), intent(in) stomach_content_mass )
```

The backend engine for calculating the stomach content mass decrement as a consequence of *digestion*. Stomach contents S(t) is emptied by a constant fraction each time step ΔS:

$$\Delta S = S_t \frac{K}{\Omega},$$

where K is the *stomach content emptify factor* parameter (commondata::stomach_content_emptify_factor) and Ω is the lifespan (commondata::lifespan parameter).

Parameters

in	*stomach_content_mass*	stomach_content_mass Current mass of the stomach contents.

Returns

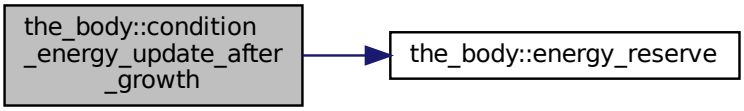
The decrement value

Definition at line 1633 of file m_body.f90.
Here is the caller graph for this function:

8.4.3.42 condition_energy_update_after_growth()

```
elemental subroutine the_body::condition_energy_update_after_growth (
            class(condition), intent(inout) this )
```

Update the energy reserves of the agent based on its current mass and length. This subroutine should be called after any event that can change the mass or/and length of the agent, e.g. food consumption.
Update the energy reserves. This is done by calling the standard function energy_reserve()
And also update the historical maximum value, if the current energy reserves value exceeds the maximum.
Definition at line 1649 of file m_body.f90.
Here is the call graph for this function:



8.4.3.43 cost_swimming_standard()

```
elemental real(srp) function the_body::cost_swimming_standard (
            class(condition), intent(in) this,
            integer, intent(in), optional steps )
```

The standard cost of swimming is a diagnostic function that shows the cost, in units of the body mass, incurred if the agent passes a distance equal to commondata::lifespan units of its body length.

Parameters

in	*steps*	steps is the optional number of steps of the agent length the agent walks. Default value is commondata::lifespan (i.e. the whole lifespan).

Returns

The cost of swimming in terms of the agent's current body mass.

Definition at line 1667 of file m_body.f90.

8.4.3.44 reproduction_cost_energy_fix()

```
elemental real(srp) function the_body::reproduction_cost_energy_fix (
            class(reproduction), intent(in) this )
```

Calculate the energetic cost of reproduction in terms of the **body mass** of the this agent. The energetic cost of reproduction is obtained as a specific fixed fraction of the current body mass of the agent defined by the `commondata::reproduction_cost_body_mass` parameter.

Returns

Returns the energetic cost of reproduction for the this agent.

Definition at line 1703 of file m_body.f90.

8.4.3.45 reproduction_cost_energy_dynamic()

```
real(srp) function the_body::reproduction_cost_energy_dynamic (
            class(reproduction), intent(in) this )
```

Calculate the energetic cost of reproduction in terms of the **body mass** of the this agent. The energetic cost of reproduction is different in males and females.

Returns

Returns the energetic cost of reproduction for the this agent.

8.4.3.45.1 Implementation details **First,** calculate the overall mass of the offspring that are produced as a result of this reproduction event μ. This is done using the procedure `reproduction::offspring_mass` (=> `the_body::reproduction_mass_offspring_calc`). The total mass of the offspring serves as a baseline for calculating the overall cost of reproduction.
Second, calculate the cost of reproduction as a sum of two components:

- component that scales with the total mass of the offspring μ;
- component that scales with the body mass of the agent M.

There are two versions of this function implemented:

- `cost_full()` where the cost component that scales with the agent's body mass is calculated from the full agent's mass *not subtracting* the total mass of the offspring:

$$C = \mu \cdot \phi + M \cdot \varphi,$$

- `cost_residual()` where the cost component that scales with the agent's body mass is calculated from the agent's *residual* body mass *after subtracting* the total mass of the offspring:

$$C = \mu \cdot \phi + (M - \mu \cdot \phi) \cdot \varphi,$$

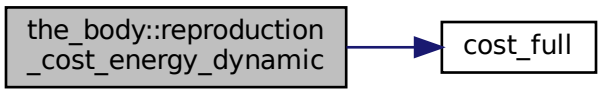
where ϕ and φ are the scaling factors that are set by the following sex-specific parameter values: Scaling factor of the offspring mass component ϕ:

- `commondata::reproduction_cost_offspring_fract_male`;
- `commondata::reproduction_cost_offspring_fract_female`.

Scaling factor of the agent's body mass component φ:

- `commondata::reproduction_cost_body_mass_factor_male`;
- `commondata::reproduction_cost_body_mass_factor_female`.

8.4.3.45.2 Notes This allows setting the cost of reproduction in a sex-specific way. For males, for example, the component proportional to the total offspring mass is set to some small value ($\phi \leq 1.0$, whereas in females, who carry the eggs, this cost is at least equal to the full offspring mass ($\phi \geq 1.0$). On the other hand, the cost component that is proportional to the agent body mass can be higher in males that in females due to competition for mates ($\varphi_{males} \geq \varphi_{females}$). Various patterns can be implemented by varying the sex-specific scaling parameters and the two versions of the backend procedure (`cost_full()`, `cost_residual()`).
Definition at line 1716 of file m_body.f90.
Here is the call graph for this function:



8.4.3.46 reproduction_cost_unsuccessful_calc()

```
real(srp) function the_body::reproduction_cost_unsuccessful_calc (
            class(reproduction), intent(in) this,
            real(srp), intent(in), optional cost_factor )
```

Calculate the costs of unsuccessful reproduction. This is calculated as a fraction of the normal cost of reproduction returned by the function `reproduction::reproduction_cost()`. Reproduction can be unsuccessful for various reasons: insufficient reserves (reproduction results in starvation death) or stochastic no success.

Parameters

in	*cost_factor*	cost_factor Optional cost factor multiplier the normal cost of reproduction is applied to. If absent, the default value set by the `commondata::reproduction_cost_unsuccess` parameter is used.

Returns

Returns the energetic cost of unsuccessful reproduction for the 'this' agent.

Unsuccessful reproduction attempt results in a cost, in terms of the body mass, that is a fraction of the normal cost of reproduction: the fraction is defined by the parameter `commondata::reproduction_cost_unsuccess` in `COMMONDATA`. The body mass of the agent is then reduced to take this fraction of the full cost of reproduction. This updated value is saved into the body mass history stack (`update_history` parameter is `TRUE`).
Definition at line 1877 of file m_body.f90.

8.4.3.47 reproduction_init_zero()

```
elemental subroutine the_body::reproduction_init_zero (
            class(reproduction), intent(inout) this )
```

Initialise the reproduction object for the agent. Everything is set to zero.
Set the total number of reproductions and offspring to zero.
Definition at line 1907 of file m_body.f90.

8.4.3.48 reproduction_n_reproductions_get()

```
elemental integer function the_body::reproduction_n_reproductions_get (
            class(reproduction), intent(in) this )
```

Get the number of reproductions for this agent.

Returns

The number of reproductions the agent had.

Definition at line 1918 of file m_body.f90.

8.4.3.49 reproduction_n_reproductions_set()

```
elemental subroutine the_body::reproduction_n_reproductions_set (
            class(reproduction), intent(inout) this,
            integer, intent(in) n_repr )
```

Set the number of reproductions for the agent.

Parameters

in	*n_repr*	n_repr The total number of reproductions for this agent.

Definition at line 1929 of file m_body.f90.

8.4.3.50 reproduction_n_offspring_get()

```
elemental integer function the_body::reproduction_n_offspring_get (
            class(reproduction), intent(in) this )
```

Get the number of offspring.

Returns

The number of offspring the agent had during its lifespan.

Definition at line 1940 of file m_body.f90.

8.4.3.51 reproduction_n_offspring_set()

```
elemental subroutine the_body::reproduction_n_offspring_set (
            class(reproduction), intent(inout) this,
            integer, intent(in) n_offspr )
```

Set the number of offspring for the agent.

Parameters

in	*n_offspr*	n_offspr The number of offspring to set for this agent.

Definition at line 1951 of file m_body.f90.

8.4.3.52 reproduction_n_increment()

```
subroutine the_body::reproduction_n_increment (
            class(reproduction), intent(inout) this,
            integer, intent(in), optional add_repr,
            real(srp), intent(in), optional average_mass_offspring )
```

Increment the number of reproductions and offspring for this agent.

Parameters

in	*add_repr*	add_repr Increment the total number of reproductions for this agent by this number; if not provided as a dummy parameter assume increment by one. **Note:** Varying the number of reproductions allows implementation of multiple fertilisations by a male, resulting in `add_repr`>1 during a single reproduction event, provided several females are present in the male's perception object. This allows modelling sexual asymmetries.
in	*average_mass_offspring*	average_mass_offspring optional average body mass of the ` offspring. If not provided, back calculated from the Fulton's condition factor and the body length of the agents at init (birth) using the `the_body::length2mass()` function.

8.4.3.52.1 Implementation details **First,** check if the `add_repr` increment parameter is provided. If not, set it to the default value = 1.
Second, increment the number of reproductions for this agent (data component, `reproduction::n_reproductions`) by the increment parameter.
Third, calculate the number of the offspring that result from this/these reproduction occurrence(s) and increment the lifetime number `reproduction::n_offspring` for the agent respectively. The number of offspring is calculated using the function `reproduction::offspring_number()` (`the_body::reproduction_n_offspring_calc`
The average mass of the offspring (`average_mass_offspring`), if provided, transfers into the above backend function.
Definition at line 1962 of file m_body.f90.

8.4.3.53 reproduction_n_offspring_calc()

```
integer function the_body::reproduction_n_offspring_calc (
            class(reproduction), intent(inout) this,
            real(srp), intent(in), optional average_mass_offspr )
```

Calculate the number of offspring per a single reproduction as a function of the agent's body mass.

Parameters

in	*average_mass_offspr*	average_mass_offspr Optional average body mass of the offspring (μ_o, see below).

Returns

Returns the number of offspring per single reproduction.

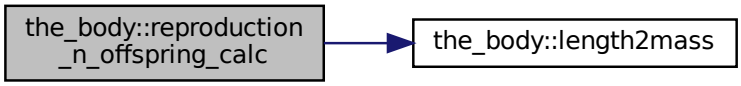
8.4.3.53.1 Implementation details Initially check if the average mass of newborn agents μ_o is provided as a dummy parameter to this function call. If not, calculate a guess of the average mass from the Fulton's condition factor and the body length parameters of the agents at init (birth) using the `the_body::length2mass()` function.
The number of offspring produced at a single reproduction scales with the body mass of the agent. **First,** all the offspring comprise the maximum combined fraction of the agent's body mass $\phi \cdot M_{agent}$, this fraction ϕ is obtained by a nonparametric relationship defined by the the interpolation grid:

- `commondata::reproduct_body_mass_offspr_abscissa`
- `commondata::reproduct_body_mass_offspr_ordinate`

The total mass of the offspring ($\phi \cdot M_{agent}$) is calculated in the procedure `reproduction::offspring_mass()` (=> `the_body::reproduction_mass_offspring_calc()`).
Second, the number of the offspring with the overall mass $\phi \cdot M_{agent}$ is calculated as the fraction:

$$\frac{\phi \cdot M_{agent}}{\mu_o},$$

where μ_o is the average mass of a single offspring at init (birth). The `floor` Fortran intrinsic function is used to calculate the integer value from this ratio. This guarantees that the number of offspring returned is the *lowest* integer value resulting from the above ratio.
Definition at line 2017 of file m_body.f90.
Here is the call graph for this function:



8.4.3.54 reproduction_ready_steroid_hormones_exceed()

```
elemental logical function the_body::reproduction_ready_steroid_hormones_exceed (
            class(reproduction), intent(in) this )
```

Determine if the agent's hormonal system is ready for reproduction.

Returns

TRUE if the agent is ready to reproduce by sufficiently high level of gonadal steroids or FALSE otherwise.

8.4.3.54.1 Implementation notes Determine if the agent's hormonal system is ready for reproduction, that is, its current level of sex steroids σ_i exceeds the baseline (initially determined by the genome) σ_0 by a factor ν determined by the parameter commondata::sex_steroids_reproduction_threshold:

$$\sigma_i > \nu\sigma_0.$$

If the level of sex steroids is insufficient, reproduction is impossible and FALSE is returned.
Definition at line 2073 of file m_body.f90.

8.4.3.55 reproduction_mass_offspring_calc()

```
real(srp) function the_body::reproduction_mass_offspring_calc (
            class(reproduction), intent(in) this )
```

Calculate the total mass of all offspring produced by this agent during a single reproduction event.

Returns

Total mass of all offspring produced by the agent during a specific reproduction event.

8.4.3.55.1 Implementation details The total mass of all offspring produced at a single reproduction scales with the body mass of the agent and is obtained by a non-parametric relationship involving non-linear interpolation. The combined offspring mass is calculated as a fraction of the agent's body mass M_{agent} using this equation:

$$\phi \cdot M_{agent},$$

where ϕ is the fraction coefficient obtained by a nonparametric relationship defined by the the interpolation grid

- `commondata::reproduct_body_mass_offspr_abscissa`
- `commondata::reproduct_body_mass_offspr_ordinate`

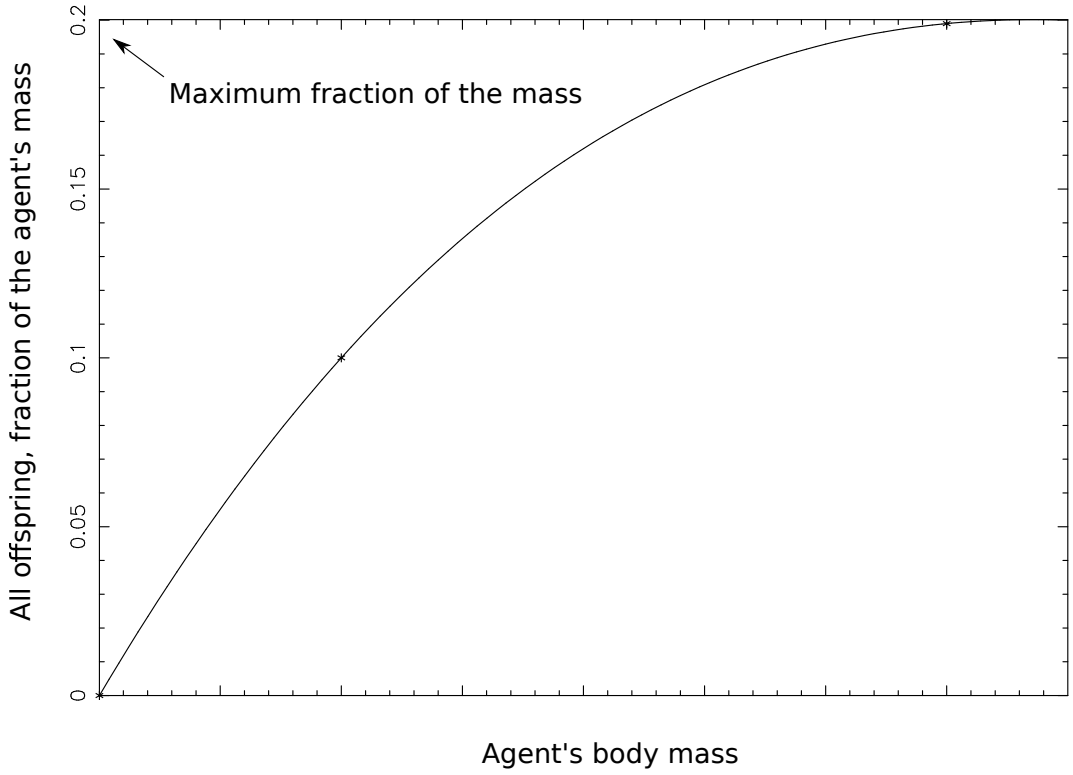


Figure 8.6 Offspring fraction of body mass

Interpolation plots can be saved in the debug mode using this plotting command: `commondata::debug_interpolate_plot_`

Warning

Involves **huge** number of plots, should normally be disabled.

Definition at line 2106 of file m_body.f90.
Here is the call graph for this function:

8.4.4 Variable Documentation

8.4.4.1 modname

```
character (len=*), parameter, private the_body::modname = "(THE_CONDITION)"  [private]
```

Definition at line 26 of file m_body.f90.

8.5 the_environment Module Reference

Definition of environmental objects.

Data Types

- type spatial

Definition of a spatial object. Spatial object determines the position of the agent, food items and other things in the simulated space. Here we use continuous 3D environment (real type coordinates)

- type spatial_moving

  Definition of a movable spatial object. It extends the the_environment::spatial object, but also adds its previous position history in stack-like arrays. The history is maintained with the commondata::add_to_history() subroutine, adding a single element on the top (end) of the history stack.

- type environment

  *Definition of the overall **environment**. Environment is a general container for all habitats, patches and other similar objects where the whole life of the agents takes place. Environment just provides the limits for all these objects.*

- type food_item

  Definition of a single food item. Food item is a spatial object that has specific location in space. It can be "created" and eaten ("disappear"). Food item is an immobile SPATIAL object that has a position in 3D space.

- type food_resource

  Definition of the super-type FOOD resource type. This is a superclass, several sub-classes can be defined for different kinds of food and prey objects.

- type predator

  *Definition of the `PREDATOR` objects. **Predator** is a moving agent that hunts on the evolving AHA agents but its internal structure is very simplistic (although we can in principle doit as a full AHA complexity with genome, GOS etc...).*

- type habitat

  *Definition of the **environment habitat** `HABITAT` object. There can potentially be of several types of habitats (patches etc.), so the superclass HABITAT defines the most general properties and procedures. More specific procedures are defined in sub-objects. Such procedures can be overriden from super-object to sub-objects providing for procedure polymorphism.*

- interface light_surface

  Calculate surface light intensity (that is subject to diel variation) for specific time step of the model. Irradiance can be stochastic if an optionallogical `stochastic` flag is set to `TRUE`.

- interface light_depth

  Calculate underwater background irradiance at specific depth.

- interface visual_range

  Calculate visual range of predator using Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`.

- interface visual_range_new

  Calculate visual range of predator using Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`.

- interface dist

  Internal distance calculation backend engine.

- interface join

  An alias for the the_environment::food_resources_collapse_global_object() method for joining food resources into a single global food resource out of the global array the_environment::global_habitats_available. See the_environment::unjoin() for how to unjoin an array of food resources back into an array.

- interface unjoin

  An alias to the_environment::food_resources_update_back_global_object() method to transfer (having been modified) food resource objects out from the single united object back to the global array the_environment::global_habitats_available. See the_environment::join() for how to join an array of food resources into a single global object.

- interface assemble

  *Interface to the procedure to **assemble** the global array of habitat objects the_environment::global_habitats_available from a list of separate habitat object components. This call.*

- interface disassemble

  *Interface to the procedure to **disassemble** the global habitats objects array the_environment::global_habitats_available back into separate habitat object components. See the_environment::global_habitats_disassemble() for the backend implementation.*

- interface operator(.cat.)

  Interface operator to concatenate two arrays of the spatial the_environment::spatial or spatial moving the_environment::spatial_moving objects.

- interface operator(.catloc.)

  *Interface operator to concatenate the **location** components of two arrays ofthe_environment::spatial **class** objects.*

- interface operator(.within.)

  Interface operators `.within.` for testing whether a spatial object (first argument lies within an environment (second argument). Usage:
- interface operator(.contains.)

  Interface operators `.contains.` for testing whether an environment object (first argument) contains a `SPATIAL` object (second argument). Usage:
- interface operator(.above.)

  Interface operators .above. for spatial objects. Usage:
- interface operator(.below.)

  Interface operators .below. for spatial objects. Usage:
- interface operator(-)

  Interface operator "-" for the the_environment::environment spatial container objects. Return an environment object that is shrunk by a fixed value in the 2D XxY plane. See `the_environment::environment_shrink_xy_fixed()`. The operator can be used as follows:

Functions/Subroutines

- elemental subroutine spatial_create_empty (this)

  These are public access functions, but probably we don't need to allow public access to functions inside generic interfaces.
- subroutine environment_whole_build_vector (this, min_coord, max_coord)

  Create the highest level container environment. Set the size of the 3D environment container as two coordinate vectors setting the minimum and maximum coordinate limits: `min_coord(1)` for x, `min_coord(2)` for y, `min_coord(3)` for z The size of the environment should be set from parameter vectors in `COMMONDATA`.
- subroutine environment_whole_build_object (this, min_coord, max_coord)

  Create the highest level container environment. Set the size of the 3D environment container as two coordinate vectors setting the minimum and maximum coordinate limits. The parameters `min_coord` and `max_coord` are SPATIAL objects.
- subroutine environment_build_unlimited (this)

  *Build an **unlimited environment**, with the spatial coordinates limited by the maximum machine supported values based on the intrinsic `huge` function.*
- type(environment) function environment_shrink_xy_fixed (this, shrink_value)

  Return an environment object that is shrunk by a fixed value in the 2D XxY plane.
- type(spatial) function environment_get_minimum_obj (this)

  *Function to get the **minimum** spatial limits (coordinates) of the environment.*
- type(spatial) function environment_get_maximum_obj (this)

  *Function to get the **maximum** spatial limits (coordinates) of the environment.*
- elemental real(srp) function environment_get_minimum_depth (this)

  *Get the **minimum depth** in this environment.*
- elemental real(srp) function environment_get_maximum_depth (this)

  *Get the **maximum depth** in this environment.*
- pure type(spatial) function, dimension(dim_environ_corners) environment_get_corners_2dxy (this, ref_depth, offset)

  Get the corners of the environment in the 2D X Y plane. This is a very simplistic procedure that works only with the box environmental objects.
- elemental logical function environment_check_located_within_3d (this, check_object)

  Check if a spatial object is actually within this environment.
- type(spatial) function environment_random_uniform_spatial_3d (this)

  *Generate a random spatial object with the uniform distribution within (i.e. bound to) **this** environment.*
- type(spatial) function environment_random_uniform_spatial_2d (this, fixdepth)

  *Generate a random spatial object with the uniform distribution within (i.e. bound to) **this** environment, the third depth coordinate is fixed.*
- type(spatial) function, dimension(num) environment_random_uniform_spatial_vec_3d (this, num)

  *Generate a vector of random spatial objects with the uniform distribution within (i.e. bound to) **this** environment.*

- type(spatial) function, dimension(size(fixdep_array)) environment_random_uniform_spatial_vec_2d (this, fixdep_array)

  *Generate a vector of random spatial objects with the uniform distribution within (i.e. bound to) **this** environment. The third, depth coordinate is non-stochastic, and provided as an array parameter.*
- type(spatial) function, dimension(num) environment_random_gaussian_spatial_3d (this, num, centroid, variance)

  *Generates a vector of random spatial object with Gaussian coordinates within (i.e. bound to) **this** environment.*
- type(spatial) function, dimension(num) environment_random_gaussian_spatial_2d (this, num, centroid, fixdepth, variance, variance_depth)

  *Generates a vector of random spatial object with Gaussian coordinates within (i.e. bound to) **this** environment. The depth coordinate is set separately and can be non-random (fixed for the whole output array) or Gaussian with separate variance.*
- subroutine environment_get_nearest_point_in_outside_obj (this, outside_object, offset_into, point_spatial, point_dist)

  Get the spatial point position within this environment that is nearest to an arbitrary spatial object located outside of the this environment. If the spatial object is actually located in this environment, return its own position.
- subroutine spatial_fix_position_3d_s (this, x, y, depth)

  Place spatial object into a 3D space, define the object's current coordinates.
- elemental subroutine spatial_fix_position_3d_o (this, location)

  Place spatial object into a 3D space, define the object's current coordinates.
- elemental subroutine spatial_make_missing (this)

  Assign all commondata::missing` coordinates to the_environment::spatial object.
- elemental type(spatial) function spatial_get_current_pos_3d_o (this)

  Get the current spatial position of a `SPATIAL` object.
- pure real(srp) function, dimension(dimensionality_default) spatial_get_current_pos_3d_v (this, vector)

  Get the current spatial position of a `SPATIAL` object.
- elemental real(srp) function spatial_get_current_pos_x_3d (this)

  Get the current `X` position of a `SPATIAL` object.
- elemental real(srp) function spatial_get_current_pos_y_3d (this)

  Get the current `Y` position of a `SPATIAL` object.
- elemental real(srp) function spatial_get_current_pos_d_3d (this)

  Get the current `DEPTH` position of a `SPATIAL` object.
- real(srp) function spatial_calc_irradiance_at_depth (this, time_step_model)

  Calculate the illumination (background irradiance) at the depth of the spatial object at an arbitrary time step of the model.
- real(srp) function spatial_visibility_visual_range_cm (this, object_area, contrast, time_step_model)

  Calculate the visibility range of a spatial object. Wrapper to the the_environment::visual_range() function. This function calculates the distance from which this object can be seen by a visual object (e.g. predator or prey).
- pure integer function spatial_get_environment_in_pos (this, environments_array)

  Identify in which environment from the input list this spatial agent is currently in. Example call:
- subroutine spatial_moving_fix_position_3d_v (this, x, y, depth)

  Place spatial movable object into a 3D space, define the object's current coordinates, but first save previous coordinates.
- elemental subroutine spatial_moving_fix_position_3d_o (this, location)

  Place spatial movable object into a 3D space, define the object's current coordinates, but first save previous coordinates.
- elemental subroutine spatial_moving_repeat_position_history_3d (this)

  Repeat (re-save) the current position into the positional history stack.
- elemental real(srp) function spatial_distance_3d (this, other)

  Calculate the Euclidean distance between two spatial objects. This is a type-bound function.
- pure type(spatial) function, dimension(:), allocatable spatial_stack2arrays (a, b)

  Concatenate two arrays of the_environment::spatial objects `a` and `b`. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.

- pure type(spatial_moving) function, dimension(:), allocatable spatial_moving_stack2arrays (a, b)

  Concatenate two arrays of the_environment::spatial_moving objects `a` and `b`. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.
- pure type(spatial) function, dimension(:), allocatable spatial_class_stack2arrays_locs (a, b)

  *Concatenate the **location** components of two arrays of the_environment::spatial class objects a and b. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.*
- elemental real(srp) function dist3d (this, other)

  *This is a **non-type-bound** version of the distance calculation function.*
- elemental real(srp) function spatial_self_distance_3d (this, from_history)

  Calculate the Euclidean distance between the current and previous position of a single spatial object.
- elemental real(srp) function spatial_moving_self_distance_3d (this, from_history)

  *Calculate the Euclidean distance between the current and previous position of a single spatial movable object. Optionally, it also calculates the total distance traversed during the `from_history` points from the history stack along with the distance from the current position and the last historical value. For example, to calculate the **average** distance throughout the whole history (`HISTORY_SIZE_SPATIAL` points) do this:*
- elemental subroutine spatial_moving_create_3d (this)

  Create a new spatial moving object. Initially it has no position, all coordinate values are `MISSING` or `INVALID` for real type coordinates.
- elemental subroutine spatial_moving_clean_hstory_3d (this)

  Create a new empty history of positions for spatial moving object. Assign all values to the MISSING value code.
- elemental subroutine spatial_moving_go_up (this, step)

  *The spatial moving object **ascends**, goes up the depth with specific fixed step size.*
- elemental subroutine spatial_moving_go_down (this, step)

  *The spatial moving object **decends**, goes down the depth with specific fixed step size.*
- subroutine spatial_moving_randomwalk_gaussian_step_3d (this, meanshift, cv_shift, environment_limits)

  Implements an optionally environment-restricted Gaussian random walk in 3D.
- subroutine spatial_moving_randomwalk_gaussian_step_25d (this, meanshift_xy, cv_shift_xy, meanshift_↩ depth, cv_shift_depth, environment_limits)

  Implements an optionally environment-restricted Gaussian random walk in a "2.5 dimensions", i.e. 2D x y with separate walk parameters for the third depth dimension.
- subroutine spatial_moving_corwalk_gaussian_step_3d (this, target, meanshift, cv_shift, is_away, ci_lim, environment_limits, is_converged, debug_reps)

  *Implements an optionally environment-restricted **correlated directional** Gaussian random walk in 3D towards (or away of) an the_environment::spatial class `target` object.*
- subroutine spatial_moving_corwalk_gaussian_step_25d (this, target, meanshift_xy, cv_shift_xy, meanshift↩ _depth, cv_shift_depth, is_away, ci_lim, environment_limits, is_converged, debug_reps)

  *Implements an optionally environment-restricted **correlated directional** Gaussian random walk in 3D towards (or away of) an the_environment::spatial class `target` object.*
- subroutine spatial_moving_dirwalk_gaussian_step_3d (this, target, meanshift, cv_shift, environment_limits)

  *Implements an optionally environment-restricted **directional** Gaussian random walk in 3D towards a `target` the_environment::spatial class object.*
- subroutine spatial_moving_dirwalk_gaussian_step_25d (this, target, meanshift_xy, cv_shift_xy, meanshift↩ _depth, cv_shift_depth, environment_limits)

  *Implements an optionally environment-restricted **directional** Gaussian random walk in "2.5"-D towards a `target` the_environment::spatial class object. i.e. 2D x y with separate walk parameters for the third depth dimension.*
- subroutine rwalk3d_array (this, dist_array, cv_array, dist_all, cv_all, environment_limits, n_walks)

  Perform one or several steps of random walk by an array of the_environment::spatial_moving class objects. This is a 3D version with the same walk parameters for the horizontal XxY plane and depth.
- subroutine rwalk25d_array (this, dist_array_xy, cv_array_xy, dist_array_depth, cv_array_depth, dist_all_xy, cv_all_xy, dist_all_depth, cv_all_depth, environment_limits, n_walks)

  Perform one or several steps of random walk by an array of the_environment::spatial_moving class objects. This is a 2.5D version with separate walk parameters for the horizontal XxY plane and depth.
- elemental logical function spatial_check_located_within_3d (this, environment_limits)

Function to check if this spatial object is located within an area set by an environmental object (parameter). This should be similar to an analogous function defined for the environment object.

- elemental logical function spatial_check_located_below (this, check_object)

  *Logical function to check if the **argument** spatial object(s) (`check_object`) is (are) located **below** the **this** reference spatial object. Elemental function that also works with arrays. Use as:*
- elemental logical function spatial_check_located_above (this, check_object)

  *Logical function to check if the **argument** spatial object(s) (`check_object`) is (are) located **above** the **this** reference spatial object. Elemental function that also works with arrays. Use as:*
- type(spatial) function spatial_get_nearest_object (this, neighbours, number)

  *Determine the nearest spatial object to **this** spatial object among an array of other spatial objects.*
- integer function spatial_get_nearest_id (this, neighbours, object)

  *Determine the nearest spatial object to **this** spatial object among an array of other spatial objects.*
- subroutine habitat_make_init (this, coord_min, coord_max, label, otherrisks, eggmortality, predators_number, loc_predators, food_abundance, loc_food, sizes_food)

  Make an instance of the habitat object (an environment superset).
- character(len=label_length) function habitat_name_get (this)

  Return the name of the habitat.
- real(srp) function habitat_get_risk_mortality (this)

  Get the mortality risk associated with this habitat.
- real(srp) function habitat_get_risk_mortality_egg (this)

  Get the egg mortality risk associated with this habitat.
- subroutine habitat_save_predators_csv (this, csv_file_name, is_success)

  Save the predators with their characteristics into a CSV file.
- subroutine save_dynamics (maxdepth, csv_file_name, is_success)

  Save diagnostics data that shows the dynamics of the light and the average depth of the food items, light at the average depth of the food items etc at each time step of the model.
- type(spatial) function environment_centre_coordinates_3d (this, nodepth)

  Determine the centroid of the environment.
- real(srp) function, private visual_range_scalar (irradiance, prey_area, prey_contrast)

  Wrapper for calculating visual range of a fish predator using the Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`. See `srgetr()` for computational details.
- real(srp) function, dimension(size(prey_area)), private visual_range_vector (irradiance, prey_area, prey_↩ contrast_vect, prey_contrast)

  Wrapper for calculating visual range of a fish predator using the Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`. See `srgetr()` for computational details.
- elemental real(srp) function visual_range_fast (irradiance, prey_area, prey_contrast)

  *Wrapper for calculating visual range of a fish predator using the Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`. This is a new **elemental** and parallel-ready visual range function wrapper making use the elemental-procedures based computational backend. See notes on `visual_range_scalar()` and `srgetr()` for computational details.*
- elemental real(srp) function, private light_surface_deterministic (tstep)

  Calculate deterministic surface light at specific time step of the model. Light (`surlig`) is calculated from a sine function. Light intensity just beneath the surface is modelled by assuming a 50 % loss by scattering at the surface:
- real(srp) function, private light_surface_stochastic_scalar (tstep, is_stochastic)

  Calculate stochastic surface light at specific time step of the model. Light (`surlig`) is calculated from a sine function. Light intensity just beneath the surface is modelled by assuming a 50 % loss by scattering at the surface:
- real(srp) function, dimension(size(tstep)), private light_surface_stochastic_vector (tstep, is_stochastic)

  Calculate stochastic surface light at specific time step of the model.
- real(srp) function, private light_depth_integer (depth, surface_light, is_stochastic)

  Calculate underwater light at specific depth given specific surface light.
- real(srp) function, private light_depth_real (depth, surface_light, is_stochastic)

  Calculate underwater light at specific depth given specific surface light.
- elemental real(srp) function dist_scalar (x1, x2, y1, y2, z1, z2)

Calculate distance between 3D or 2D points. This is a function engine for use within type bound procedures. ***Example*** *(dist_scalar):*

- pure real(srp) function dist_vector_nd (cvector1, cvector2)

  Calculate distance between N-dimensional points. This is a function engine for use within other type bound procedures.
- pure real(srp) function dist2_vector (cvector1, cvector2)

  Calculate the squared distance between two N-dimensional points.
- pure real(srp) function vect_magnitude (vector)

  Calculate the magnitude of an arbitrary N-dimensional vector. This is a raw vector backend.
- elemental real(srp) function dist2step (average_distance, dimensionality)

  Calculate the unit step along a single coordinate axis given the average distance between any two points in a N-dimensional Gaussian random walk.
- elemental subroutine food_item_create (this)

  Create a single food item at an undefined position with default size.
- elemental subroutine food_item_make (this, location, size, iid)

  Make a single food item, i.e. place it into a specific position in the model environment space and set the size.
- logical function food_item_capture_success_stochast (this, prob)

  Stochastic outcome of ***this*** *food item capture by an agent. Returns TRUE if the food item is captured.*
- real(srp) function food_item_capture_probability_calc (this, distance, time_step_model)

  Calculate the probability of capture of ***this*** *food item by a predator agent depending on the distance between the agent and this food item.*
- real(srp) function food_item_visibility_visual_range (this, object_area, contrast, time_step_model)

  Calculate the visibility range of this food item. Wrapper to the the_environment::visual_range() function. This function calculates the distance from which this food item can be seen by a predator (i.e. the default predator's visual range).
- real(srp) function minimum_depth_visibility (target_range, depth_range_min, depth_range_max, object_↩ area, object_contrast, time_step_model)

  Find the depth at which the visibility of a spatial object becomes smaller than a specific distance value `target_↩ range`*.*
- elemental subroutine food_item_disappear (this)

  Make the food item "disappear" and take the "eaten" state, i.e. impossible for consumption by the agents.
- elemental logical function food_item_is_eaten_unavailable (this)

  Logical check-indicator function for the food item being eaten and not available.
- elemental logical function food_item_is_available (this)

  Logical check-indicator function for the food item being available.
- elemental real(srp) function food_item_get_size (this)

  Get the size component of the food item object.
- elemental real(srp) function size2mass_food (radius)

  Calculate the mass of a food item, the non-OO backend.
- elemental real(srp) function mass2size_food (mass)

  Calculate the size (radius) of a food item, a reverse function of the_environment::size2mass_food():
- elemental real(srp) function food_item_get_mass (this)

  Calculate and get the mass of the food item.
- elemental subroutine food_item_set_iid (this, iid)

  Set unique id for the food item object.
- elemental subroutine food_item_clone_assign (this, the_other)

  Clone the properties of this food item to another food item.
- elemental integer function food_item_get_iid (this)

  Get the unique id of the food item object.
- pure subroutine food_resource_make (this, label, abundance, locations, sizes)

  Make food resource object. This class standard constructor.
- elemental integer function food_resource_get_abundance (this)

  Get the number of food items in the food resource.

- elemental character(len=label_length) function food_resource_get_label (this)

  Get the label of the this food resource.
- pure subroutine food_resource_destroy_deallocate (this)

  Delete and deallocate food resource object. This class standard destructor.
- pure type(spatial) function, dimension(size(this%food)) food_resource_locate_3d (this)

  Get the location object array (array of `SPATIAL` objects) of a food resource object.
- real(srp) function food_resource_calc_average_distance_items (this, n_sample)

  Calculate the average distance between food items within a resource. e.g. to compare it with the agent's random walk step size.
- subroutine food_resource_replenish_food_items_all (this, replace)

  Replenish and restore food resource. The food resource is replenished by substituting randomly selected `replace` food items or all items if `replace` is omitted or exceeds the actual number of food items. Unlike the the_environment::food_resource::make() method, the sizes and the positions of the food items within the resource are reused from the previous positions (previously explicitly set set by the the_environment::food_resource::make() method).
- subroutine food_resource_migrate_move_items (this, max_depth, time_step_model)

  This subroutine implements the migration of all the food items in the resource according to the plankton migration pattern from the G1 model (HED18). Briefly, the movement of each of the food items has two components:
- subroutine food_resource_rwalk_items_default (this)

  Perform a random walk step for all food items within the food resource. The walk is performed with the default parameters:
- subroutine migrate_food_vertical (habitats, time_step_model)

  Migrate food items in a whole array of food resources. The array is normally the the_environment::global_habitats_available.
- subroutine rwalk_food_step (habitats)

  Perform a random walk of food items in a whole array of food resources. The array is normally the the_environment::global_habitats_available. This procedure is a wrapper for the_environment::food_resource::rwalk() to do a walk on a whole array of habitats and linked food resources.
- real(srp) function center_depth_sinusoidal (tstep, depth)

  This function calculates the target depth for the sinusoidal vertical migration pattern of the food items at each time step of the model. See the_environment::food_resource_migrate_move_items() for the calling procedure.
- subroutine food_resource_save_foods_csv (this, csv_file_name, is_success)

  Save characteristics of food items in the resource into a CSV file.
- elemental subroutine food_resource_sort_by_size (this, reindex)

  Sort the food resource objects within the array by their sizes. The two subroutines below are a variant of the recursive quick-sort algorithm adapted for sorting real components of the the `FOOD_RESOURCE` object.
- pure subroutine food_resource_reset_iid_all (this, start_iid)

  Reset individual iid for the food resource. Individual iids must normally coincide with the array order index. If it is changed after sorting, iids no longer reflect the correct index. So this subroutine resets iids to be coinciding with each food item index.
- subroutine reindex_food_resources (resource_1, resource_2, resource_3, resource_4, resource_↩ 5, resource_6, resource_7, resource_8, resource_9, resource_10, resource_11, resource_12, resource_13, resource_14, resource_15, resource_16, resource_17, resource_18, resource_19, resource_20)

  Reset and reindex iids for an input list of several food resources. As the result of this subroutine all food items across all the resources within the whole list will have unique iids.
- subroutine food_resources_collapse (food_resource_collapsed, resource_1, resource_2, resource_3, resource_4, resource_5, resource_6, resource_7, resource_8, resource_9, resource_10, resource_11, resource_12, resource_13, resource_14, resource_15, resource_16, resource_17, resource_18, resource↩ _19, resource_20, reindex, label)

  Collapse several food resources into one. The collapsed resource can then go into the perception system. The properties of the component resources are retained in the collapsed resource.
- type(food_resource) function food_resources_collapse_global_object (reindex, label)

  Join food resources into a single global food resource out of the global array the_environment::global_habitats_available. See the_environment::unjoin() for how to unjoin an array of food resources back into an array. The joined resource can then go into the perception system. The properties of the component resources are retained in the collapsed resource.

- subroutine food_resources_update_back (food_resource_collapsed, resource_1, resource_2, resource_↩ 3, resource_4, resource_5, resource_6, resource_7, resource_8, resource_9, resource_10, resource_11, resource_12, resource_13, resource_14, resource_15, resource_16, resource_17, resource_18, resource↩ _19, resource_20, reindex)

  Transfer back the resulting food resources into their original objects out from a collapsed object from `food_↩ resources_collapse`.
- subroutine food_resources_update_back_global_object (food_resource_collapsed, reindex)

  Transfer the (having been modified) food resource objects from the single united object `food_resource_↩ collapsed` *back to the global array the_environment::global_habitats_available array. See the_environment::join() for how to join an array of food resources into a single global object.*
- subroutine global_habitats_assemble (habitat_1, habitat_2, habitat_3, habitat_4, habitat_5, habitat_↩ 6, habitat_7, habitat_8, habitat_9, habitat_10, habitat_11, habitat_12, habitat_13, habitat_14, habitat_15, habitat_16, habitat_17, habitat_18, habitat_19, habitat_20, reindex)

  Assemble the global habitats objects array the_environment::global_habitats_available from a list of separate habitat objects. This call.
- subroutine global_habitats_disassemble (habitat_1, habitat_2, habitat_3, habitat_4, habitat_5, habitat_↩ 6, habitat_7, habitat_8, habitat_9, habitat_10, habitat_11, habitat_12, habitat_13, habitat_14, habitat_15, habitat_16, habitat_17, habitat_18, habitat_19, habitat_20, reindex)

  Disassemble the global habitats objects array the_environment::global_habitats_available into separate habitat object.
- subroutine spatial_neighbours_distances (this, neighbours, dist, index_vector, ranks, rank_max, error_flag)

  Calculate the distances between ***this*** *spatial object and an array of its neighbours. Optionally output the distances, sorting index vector and rankings vector for each of these neighbours. Optionally do only partial indexing, up to the order* `rank_max` *for computational speed. Procedure* `ARRAY_INDEX()` *from HEDTOOLS is used as the computational backend for partial segmented indexing.*
- elemental subroutine predator_make_init (this, body_size, attack_rate, position, label)

  Initialise a predator object.
- subroutine predator_label_set (this, label)

  Set label for the predator, if not provided, set it random.
- elemental real(srp) function predator_get_body_size (this)

  Accessor function for the predator body size (length).
- elemental real(srp) function predator_get_attack_rate (this)

  Accessor function for the predator attack rate.
- real(srp) function predator_capture_risk_calculate_fish (this, prey_spatial, prey_length, prey_distance, is_↩ freezing, time_step_model, debug_plot_file)

  Calculates the risk of capture of the fish with the spatial location defined by `prey_spatial` *and the body length equal to* `prey_length`. *This is a backend function bound to the predator rather than prey object.*
- subroutine predator_capture_risk_calculate_fish_group (this, prey_spatial, prey_length, is_freezing, time_↩ step_model, risk, risk_indexed, index_dist)

  Calculates the risk of capture by a specific predator of an array of the fish agents with the spatial locations array defined by `prey_spatial` *and the body length array* `prey_length`. *This subroutine takes account of both the predator dilution and confusion effects and risk adjusted by the distance towards the predator.*
- real(srp) function predator_visibility_visual_range (this, object_area, contrast, time_step_model)

  Calculate the visibility range of this predator. Wrapper to the the_environment::visual_range() function. This function calculates the distance from which this predator can be seen by a visual object (e.g. the agent).
- real(srp) function distance_average (spatial_objects, sample_size)

  Calculates the average nearest neighbour distance amongst an array of spatial objects (class) by sampling `sample_size` *of them. The sample size* `sample_size` *is optional, if not provided set to* `SAMPLE_SIZE↩ _DEFAULT=25`.

Visual range calculation backend.

The subroutines the_environment::srgetr(), the_environment::easyr() and the_environment::deriv() should be better isolated into a separate module or form a submodule, but it is not used here as submodules are a F2008 feature not supported by all compiler systems. Anyway, submodule is not essential here.

Note

*Note that all these backend procedures are now **pure** and therefore parallel-safe.*

- elemental subroutine, private srgetr (r, c, C0, Ap, Vc, Ke, Eb, IER)

  Obtain visual range by solving the non-linear equation by means of Newton-Raphson iteration and derivation in subroutine the_environment::deriv(). Initial value is calculated in the_environment::easyr(). The calculation is based on the model described in Aksnes & Utne (1997) Sarsia 83:137-147.
- elemental subroutine, private easyr (r, C0, Ap, Vc, Ke, Eb)

  Obtain a first estimate of visual range by using a simplified expression of visual range. See `srgetr()` for more details.
- elemental subroutine, private deriv (r, F1, FDER, c, C0, Ap, Vc, Ke, Eb)

  Derivation of equation for visual range of a predator. See the_environment::srgetr() for more details.

Computational geometry backend: polygon2D

Rudimentary computational geometry (geo_) procedures, based on 2D polygons (poly2d) with fixed depth or depth ignored.

Note

Manually constructed using 2D X x Y vectors ignoring the depth dimension.

- subroutine geo_poly2d_dist_point_to_section (point, sectp1, sectp2, min_dist, point_segment)

  Calculates the minimum distance from a the_environment::spatial class object to a line segment delimited by two the_environment::spatial class endpoints in the 2D XY plane (the depth coordinate is ignored). (The algorithm is partially based on `this`.)
- subroutine geo_poly3d_dist_point_to_section (point, sectp1, sectp2, min_dist, point_segment)

  Calculates the minimum distance from a the_environment::spatial class object to a line segment delimited by two the_environment::spatial class endpoints in the 3D XY space. (The algorithm is partially based on `this`.)
- type(spatial) function offset_dist (obj_a, obj_b, offset)

  Calculate a the_environment::spatial target with an offset.

Variables

- character(len= ∗), parameter, private modname = "(THE_ENVIRONMENT)"
- integer, parameter, private dimensionality_default = 3

  Default dimensionality of the environment universe.
- integer, parameter dim_environ_corners = 4

  The number of corners for an environment object in the 2D X∗x∗Y plane.
- type(habitat), dimension(:), allocatable, public global_habitats_available

  A list (array) of all the the_environment::habitat objects available to the agents. This single array should encompass all the locations that the agent can potentially be in (e.g. migrate from one to another).

8.5.1 Detailed Description

Definition of environmental objects.

8.5.2 THE_ENVIRONMENT module

This module defines various environment objects and primitives, starting from the basic primitive a spatial object the_environment::spatial. This object represents a point in a three-dimensional space. The agent class hierarchy starts from this spatial primitive as the agent is a spatial object.

8.5.3 Function/Subroutine Documentation

8.5.3.1 spatial_create_empty()

```
elemental subroutine the_environment::spatial_create_empty (
            class(spatial), intent(inout) this )
```

These are public access functions, but probably we don't need to allow public access to functions inside generic interfaces.
We do not need specific functions outside of this module, always use generic functions. Create an empty spatial object. The object's starting coordinates get all `MISSING` values.
Definition at line 939 of file m_env.f90.

8.5.3.2 environment_whole_build_vector()

```
subroutine the_environment::environment_whole_build_vector (
            class(environment), intent(inout) this,
            real(srp), dimension(3), intent(in) min_coord,
            real(srp), dimension(3), intent(in) max_coord )
```

Create the highest level container environment. Set the size of the 3D environment container as two coordinate vectors setting the minimum and maximum coordinate limits: `min_coord(1)` for *x*, `min_coord(2)` for *y*, `min_coord(3)` for *z* The size of the environment should be set from parameter vectors in `COMMONDATA`.

Parameters

min_coord	Minimum coordinate bound for the environment.
max_coord	Maximum coordinate bound for the environment.

Note

This version accepts simple *arrays* as the environment coordinates.

Warning

Not-extensible version. TODO: Do we need it? Deprecate? There is a generic function `build` that should normally be used.

Definition at line 958 of file m_env.f90.

8.5.3.3 environment_whole_build_object()

```
subroutine the_environment::environment_whole_build_object (
            class(environment), intent(inout) this,
            type(spatial), intent(in) min_coord,
            type(spatial), intent(in) max_coord )
```

Create the highest level container environment. Set the size of the 3D environment container as two coordinate vectors setting the minimum and maximum coordinate limits. The parameters `min_coord` and `max_coord` are SPATIAL objects.

Parameters

min_coord	Minimum coordinate bound for the environment, `SPATIAL` object.
max_coord	Maximum coordinate bound for the environment, `SPATIAL` object.

Note

This version accepts `SPATIAL` *objects* as the environment coordinates.

Definition at line 988 of file m_env.f90.

8.5.3.4 environment_build_unlimited()

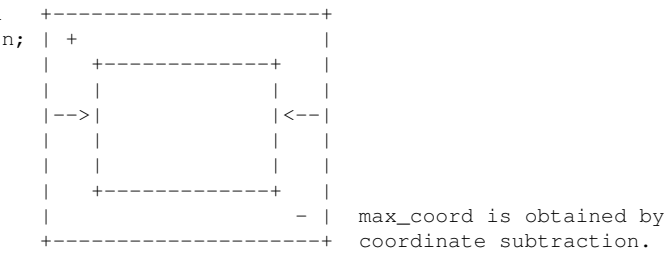
```
subroutine the_environment::environment_build_unlimited (
            class(environment), intent(inout) this )
```

Build an **unlimited environment**, with the spatial coordinates limited by the maximum machine supported values based on the intrinsic `huge` function.
Definition at line 1005 of file m_env.f90.

8.5.3.5 environment_shrink_xy_fixed()

```
type(environment) function the_environment::environment_shrink_xy_fixed (
            class(environment), intent(in) this,
            real(srp), intent(in) shrink_value )
```

Return an environment object that is shrunk by a fixed value in the 2D XxY plane.
Here is an illustration of the function. The outer box is the input environment, the inner box is the shrunken environment that is returned. The shrinkage value is fixed, defined by the second function parameter. The depth is ignored in this function working with the simplistic box-like environment objects.



There is a user defined operator − (minus), that can be used as follows:
`temp_hab = habitat_safe - 0.5_srp`

Warning

Valid only for the simplistic box-like environments; should be reimplemented if the environment is implemented as an arbitrary polyhedron.

Definition at line 1047 of file m_env.f90.

8.5.3.6 environment_get_minimum_obj()

```
type(spatial) function the_environment::environment_get_minimum_obj (
            class(environment), intent(in) this )
```

Function to get the **minimum** spatial limits (coordinates) of the environment.

Returns

The minimum spatial bound of the environment, as a `SPATIAL` object.

Definition at line 1066 of file m_env.f90.

8.5.3.7 environment_get_maximum_obj()

```
type(spatial) function the_environment::environment_get_maximum_obj (
            class(environment), intent(in) this )
```

Function to get the **maximum** spatial limits (coordinates) of the environment.

Returns

The maximum spatial bound of the environment, as a `SPATIAL` object.

Definition at line 1082 of file m_env.f90.

8.5.3.8 environment_get_minimum_depth()

```
elemental real(srp) function the_environment::environment_get_minimum_depth (
            class(environment), intent(in) this )
```

Get the **minimum depth** in this environment.

Returns

The maximum depth of this environment

Definition at line 1095 of file m_env.f90.

8.5.3.9 environment_get_maximum_depth()

```
elemental real(srp) function the_environment::environment_get_maximum_depth (
            class(environment), intent(in) this )
```

Get the **maximum depth** in this environment.

Returns

The maximum depth of this environment

Definition at line 1106 of file m_env.f90.

8.5.3.10 environment_get_corners_2dxy()

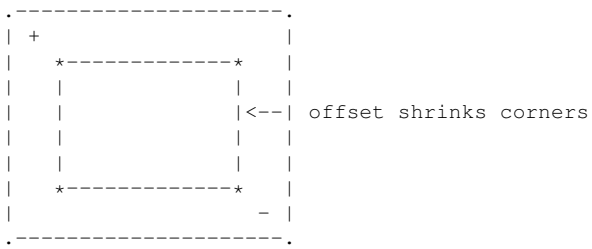
```
pure type(spatial) function, dimension(dim_environ_corners) the_environment::environment_get↩
_corners_2dxy (
            class(environment), intent(in) this,
            real(srp), intent(in), optional ref_depth,
            real(srp), intent(in), optional offset )
```

Get the corners of the environment in the 2D X Y plane. This is a very simplistic procedure that works only with the box environmental objects.

Warning

Should be reimplemented if the environment is implemented as an arbitrary polyhedron.

Parameters

in	*ref_depth*	ref_depth optional parameter setting the fixed depth for the returned corner objects. If not provided, a fixed default value equal to the local parameter `REF_DEPTH_DEF=0.0` is used.
in	*offset*	offset The offset that can be set to make the borders and corners of the environment inside at a fixed value. If offset is absent, the actual corners of the environment are returned. The offset parameter works as the `shrink2d` function (the_environment::environment::shrink2d()`. 

Returns

Returns an array of the environment corners in the 2D *X* x *Y* plane. The number of corners for the environment object in the 2D *X* x *Y* plane is defined by the parameter constant the_environment::dim_environ_corners.

8.5.3.10.1 Implementation details The corners 1,2,3,4 of the simplistic box-like environmental object are defined as follows:

```
(1:minX,minY) ----- (2:maxX,minY)
 |                           |
(4:minX,maxY) ----- (3:maxX,maxY)
```

These four points are returned as the the_environment::spatial class objects.

Warning

The order of the points is important because it is also used in the `nearest_target` procedure the_environment::environment_get_nearest_point_in_outside_obj().

Definition at line 1120 of file m_env.f90.

8.5.3.11 environment_check_located_within_3d()

```
elemental logical function the_environment::environment_check_located_within_3d (
             class(environment), intent(in) this,
             class(spatial), intent(in) check_object )
```

Check if a spatial object is actually within this environment.

Returns

TRUE if the spatial object is located within the environment.

Parameters

check_object	A spatial object (`SPATIAL` or any **extension**) to check. There is a user-defined operator: `if ( environment .contains. object ) then`

Definition at line 1207 of file m_env.f90.

8.5.3.12 environment_random_uniform_spatial_3d()

```
type(spatial) function the_environment::environment_random_uniform_spatial_3d (
             class(environment), intent(in) this )
```

Generate a random spatial object with the uniform distribution within (i.e. bound to) **this** environment.

Returns

uniformly distributed random `SPATIAL` object bound to `this` environment.

Definition at line 1239 of file m_env.f90.

8.5.3.13 environment_random_uniform_spatial_2d()

```
type(spatial) function the_environment::environment_random_uniform_spatial_2d (
             class(environment), intent(in) this,
             real(srp), intent(in) fixdepth )
```

Generate a random spatial object with the uniform distribution within (i.e. bound to) **this** environment, the third depth coordinate is fixed.

Returns

uniformly distributed random SPATIAL object bound to this environment.

Definition at line 1261 of file m_env.f90.

8.5.3.14 environment_random_uniform_spatial_vec_3d()

```
type(spatial) function, dimension(num) the_environment::environment_random_uniform_spatial_↩
vec_3d (
            class(environment), intent(in) this,
            integer, intent(in) num )
```

Generate a vector of random spatial objects with the uniform distribution within (i.e. bound to) **this** environment.

Parameters

the	dimension(size) of the vector to generate

Returns

uniformly distributed random SPATIAL object bound to this environment.

Warning

Intel Fortran porting note. This whole array function **does not work under Intel Fortran 13**, issues *stack overflow* runtime error, although compiles without issues: `loc_food_here = thisuniform(thisfoodnumber↩_food_items).`

Definition at line 1290 of file m_env.f90.

8.5.3.15 environment_random_uniform_spatial_vec_2d()

```
type(spatial) function, dimension(size(fixdep_array)) the_environment::environment_random_↩
uniform_spatial_vec_2d (
            class(environment), intent(in) this,
            real(srp), dimension(:), intent(in) fixdep_array )
```

Generate a vector of random spatial objects with the uniform distribution within (i.e. bound to) **this** environment. The third, depth coordinate is non-stochastic, and provided as an array parameter.

Parameters

the	dimension(size) of the vector to generate

Returns

uniformly distributed random SPATIAL object bound to this environment.

8.5.3.15.1 Implementation details We call the type bound `uniform_s=>environment_random_↩uniform_spatial_3d` to get a vector of uniformly-distributed spatial objects.
Definition at line 1331 of file m_env.f90.

8.5.3.16 environment_random_gaussian_spatial_3d()

```
type(spatial) function, dimension(num) the_environment::environment_random_gaussian_spatial_3d
(
            class(environment), intent(in) this,
```

```
integer, intent(in) num,
class(spatial), intent(in), optional centroid,
real(srp), intent(in), optional variance )
```

Generates a vector of random spatial object with Gaussian coordinates within (i.e. bound to) **this** environment.

Parameters

num	the dimension(size) of the vector to generate
centroid	Optional centroid of the Gaussian scatter. If not provided, will select random uniformly distributed point.
variance	Gaussian variance parameter.

Returns

Gaussian ranrom SPATIAL object bound to `this` environment.

8.5.3.16.1 Implementation details First, check optional centroid and set local centroid.
The centroid that is provided is accepted only if it is within **this** environment.
If a centroid is provided but is outside of the environment, reset it to random uniform.
Now, generate Gaussian spatial objects and fill the output array.
First, generate random Gaussian coordinanes for a temporary spatial object.
Make sure this spatial object is within the bounding environment.
Finally, set assign the output array component to this object.
Definition at line 1370 of file m_env.f90.

8.5.3.17 environment_random_gaussian_spatial_2d()

```
type(spatial) function, dimension(num) the_environment::environment_random_gaussian_spatial_2d
(
              class(environment), intent(in) this,
              integer, intent(in) num,
              class(spatial), intent(in), optional centroid,
              real(srp), intent(in), optional fixdepth,
              real(srp), intent(in), optional variance,
              real(srp), intent(in), optional variance_depth )
```

Generates a vector of random spatial object with Gaussian coordinates within (i.e. bound to) **this** environment. The depth coordinate is set separately and can be non-random (fixed for the whole output array) or Gaussian with separate variance.

Parameters

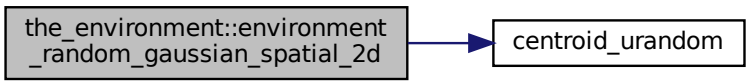
num	the dimension(size) of the vector to generate
centroid	Optional centroid of the Gaussian scatter. If not provided, will select random uniformly distributed point.
fixdepth	Optional fixed depth for the generated Gaussian objects.
variance	Gaussian variance parameter.
variance_depth	Gaussian variance parameter for the fixed depth.

Returns

Gaussian random SPATIAL object bound to `this` environment.

8.5.3.17.1 Implementation details First, check optional centroid parameter and set local centroid.
Make sure fixdepth is within the allowed environmental range.
If fixed depth is provided, we use the centroid but take the depth from the `fixdepth` parameter.
And check if the resulting centroid is within this environment. If not, make it random uniform making use of the fixed depth.

If the fixdepth is not conformant with this environment, use centroid provided and discard the fixdepth parameter
And check if the resulting centroid is within this environment, if not, discard both the centroid and the fixed depth parameters and use random uniform centroid as the last resort.
If fixed depth is not provided, however, we use the fixed depth from the centroid
The centroid that is provided is accepted only if it is within **this** environment.
If a centroid is provided but is outside of the environment, reset it to random uniform.
Check if it's conformant with the environment. Use if if okay.
If the depth provided is not conformant, discard and use random.
Finally, if neither centroid nor depth is provided, use random uniform.
Check if separate variance parameter for the depth is provided. This sets if **stochastic depth** is to be generated.
If separate depth variance parameter is provided, depth is stochastic Gaussian. Generate Gaussian spatial objects and fill the output array.
First, generate random Gaussian coordinanes for a temporary spatial object.
Finally, set assign the output array component to this object.
If there is no separate variance parameter for the depth, **depth fixed** deterministic, identical in for the whole array.
Set the fixed depth.
Now, generate Gaussian spatial objects and fill the output array.
First, generate random Gaussian coordinanes for a temporary spatial object.
Finally, set assign the output array component to this object.
Definition at line 1450 of file m_env.f90.
Here is the call graph for this function:



8.5.3.18 environment_get_nearest_point_in_outside_obj()

```
subroutine the_environment::environment_get_nearest_point_in_outside_obj (
             class(environment), intent(in) this,
             class(spatial), intent(in) outside_object,
             real(srp), intent(in), optional offset_into,
             type(spatial), intent(out), optional point_spatial,
             real(srp), intent(out), optional point_dist )
```

Get the spatial point position within this environment that is nearest to an arbitrary spatial object located outside of the this environment. If the spatial object is actually located in this environment, return its own position.

Note

This function is necessary for the implementation of migration behaviour across two or several environments or habitats: it allows to set the (nearest) target point in the desired target environment.

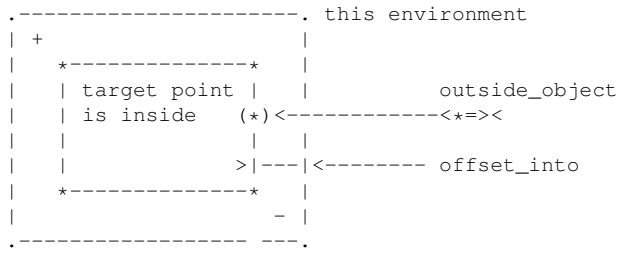
Warning

Valid only for the simplistic box-like environments; should be reimplemented if the environment is implemented as an arbitrary polyhedron.

Parameters

in	*outside_object*	outside_object is the outside object, the minimum distances to the environment and the closest point are evaluated for this object.

Parameters

in	*offset_into*	offset_into optional offset guaranteeing that the nearest point is located more or less deeply within the this target environment rather than just at the border of the environment. This parameter is useful if the nearest point in the environment actually sets the target point, to which the agent represented by the outside_object spatial is relocating. In such a case, the agent is therefore moving into the environment, at a distance offset_into rather than just to the edge of this target environment. This is illustrated by the below: 
out	*point_spatial*	

Returns

The point within the this environment that is the nearest to the outside_object the_environment::spatial class target object.

8.5.3.18.1 Implementation notes First, check if the outside object is actually located within the target environment.

- If so, the distance between the object and the environment is zero. This is a degenerate case that is treated separately: the nearest point within the environment coincides with the location of the outside object itself.

If the object is indeed outside, determine the four corners of the this environment. These corners are delimiting the outside of the this environment. The reference depth is set to the depth of the outside object.
@nolte Note that if the offset_into offset parameter is set, the corners are adjusted to this offset value, so that they are actually inside the this environment object.
Calculate the distances and the nearest points between the outside object and the four outer segments (1,2), (2,3), (3,4), (4,1).

Warning

This fast but **unsafe** implementation requires **the same order of points** to be set also in the the_environment::environment_get_corners_2dxy(). For arbitrary order of points, a full cross loop is needed, with the sizes of the segment_nearest_obj segment_distance arrays set to the square DIM↩_ENVIRON_CORNERS * DIM_ENVIRON_CORNERS.

```
1, 2              1------------2
2, 3              |            |
3, 4              |            |
4, 1              4------------3
```

Finally, the returned nearest point is the point where the distance between the outside point and any of the outer segments of the environment reaches the minimum value.
Definition at line 1632 of file m_env.f90.

8.5.3.19 spatial_fix_position_3d_s()

```
subroutine the_environment::spatial_fix_position_3d_s (
             class(spatial), intent(inout) this,
             real(srp), intent(in) x,
             real(srp), intent(in) y,
             real(srp), intent(in) depth )
```

Place spatial object into a 3D space, define the object's current coordinates.

Parameters

coordinates	The spatial objects will now get these coordinates.

Note

This is actually the **constructor** for the SPATIAL object giving it its value and existence.

This version takes *scalar coordinates* as the argument.

Warning

This implementation is **not extensible**.

Definition at line 1756 of file m_env.f90.

8.5.3.20 spatial_fix_position_3d_o()

```
elemental subroutine the_environment::spatial_fix_position_3d_o (
            class(spatial), intent(inout) this,
            type(spatial), intent(in) location )
```

Place spatial object into a 3D space, define the object's current coordinates.

Parameters

location	SPATIAL location that will be assigned to the current spatial object.

Note

This is actually the **constructor** for the SPATIAL object giving it its value and existence.

This version takes a SPATIAL object as argument.

Definition at line 1776 of file m_env.f90.

8.5.3.21 spatial_make_missing()

```
elemental subroutine the_environment::spatial_make_missing (
            class(spatial), intent(inout) this )
```

Assign all commondata::missing` coordinates to the_environment::spatial object.
Definition at line 1792 of file m_env.f90.

8.5.3.22 spatial_get_current_pos_3d_o()

```
elemental type(spatial) function the_environment::spatial_get_current_pos_3d_o (
            class(spatial), intent(in) this )
```

Get the current spatial position of a SPATIAL object.

Returns

Current spatial coordinates as a SPATIAL object.

Note

This function returns **SPATIAL** type object (3D coordinates).

This is a standard extensible version.

Definition at line 1806 of file m_env.f90.

8.5.3.23 spatial_get_current_pos_3d_v()

```
pure real(srp) function, dimension(dimensionality_default) the_environment::spatial_get_↩
current_pos_3d_v (
             class(spatial), intent(in) this,
             logical, intent(in) vector )
```

Get the current spatial position of a SPATIAL object.

Parameters

vector	flag indicating if we return a 3D vector rather than SPATIAL type object. **Note:** We need this logical parameter to avoid ambiguity in calling the generic function: 'Error: 'spatial_get_current_pos_3d' and 'spatial_get_current_pos_3d_v' for GENERIC 'now' at (1) are ambiguous`. The parameter itself is **not used**. So, for vector-output must always be TRUE.

Returns

Current spatial coordinates as a 3-dimensional array.

Note

This function returns **array** of 3D coordinates.

Warning

This function is **non extensible**.

Note

The *vector form* of the location function is particularly convenient for data output into the LOGGER module that does not accept object data, but does accept simple array data, e.g.:

```
call LOG_DBG ("location=" //                                &
         tostr(proto_parents%individual(ind)%location(.true.)))
```

Definition at line 1836 of file m_env.f90.

8.5.3.24 spatial_get_current_pos_x_3d()

```
elemental real(srp) function the_environment::spatial_get_current_pos_x_3d (
             class(spatial), intent(in) this )
```

Get the current X position of a SPATIAL object.

Returns

x_pos current X coordinate of the SPATIAL object.

Note

Not sure if really much needed.

Definition at line 1859 of file m_env.f90.

8.5.3.25 spatial_get_current_pos_y_3d()

```
elemental real(srp) function the_environment::spatial_get_current_pos_y_3d (
             class(spatial), intent(in) this )
```

Get the current Y position of a SPATIAL object.

Returns

x_pos current X coordinate of the SPATIAL object.

Note

Not sure if really much needed.

Definition at line 1872 of file m_env.f90.

8.5.3.26 spatial_get_current_pos_d_3d()

```
elemental real(srp) function the_environment::spatial_get_current_pos_d_3d (
            class(spatial), intent(in) this )
```

Get the current DEPTH position of a SPATIAL object.

Returns

x_pos current X coordinate of the SPATIAL object.

Note

Not sure if really much needed.

Definition at line 1885 of file m_env.f90.

8.5.3.27 spatial_calc_irradiance_at_depth()

```
real(srp) function the_environment::spatial_calc_irradiance_at_depth (
            class(spatial), intent(in) this,
            integer, intent(in), optional time_step_model )
```

Calculate the illumination (background irradiance) at the depth of the spatial object at an arbitrary time step of the model.

Warning

Cannot implement a generic function accepting also vectors of this objects as only elemental object-bound array functions are allowed by the standard. This function cannot be elemental, so passed-object dummy argument must always be scalar.

8.5.3.27.1 Implementation details Check optional time step parameter. If unset, use global commondata::global_time_s
Calculate ambient illumination / irradiance at the depth of this food item at the given time step.
Definition at line 1901 of file m_env.f90.

8.5.3.28 spatial_visibility_visual_range_cm()

```
real(srp) function the_environment::spatial_visibility_visual_range_cm (
            class(spatial), intent(in) this,
            real(srp), intent(in), optional object_area,
            real(srp), intent(in), optional contrast,
            integer, intent(in), optional time_step_model )
```

Calculate the visibility range of a spatial object. Wrapper to the the_environment::visual_range() function. This function calculates the distance from which this object can be seen by a visual object (e.g. predator or prey).

Warning

Cannot implement a generic function accepting also vectors of this objects as only elemental object-bound array functions are allowed by the standard. This function cannot be elemental, so passed-object dummy argument must always be scalar.

Parameters

in	*object_area*	object_area is the **mandatory** area of the spatial object (m).

Note

object_area has optional attribute here with the base the_environemnt::spatial class object can be only optional because it is optional in all extension classes (the_environment::food_item, the_environment::predator, the_body::condition,the_neurobio::spatialobj_percept_comp). However, it is actually **mandatory** here and not providing object size results in wrong calculations. Such a case is logged with the commondata::ltag_error logger tag.

Parameters

in	*contrast*	contrast is optional inherent contrast of this spatial object. the default contrast of all objects is defined by the commondata::preycontrast_default parameter.
in	*time_step_model*	optional time step of the model, if absent gets the current time step as defined by the value of `commondata::global_time_step_model_current`.

Returns

The maximum distance (m) from which this object can be seen.

PROCNAME is the procedure name for logging and debugging

8.5.3.28.1 Implementation details Check if optional object area parameter is present. For a base the_environment::spatial object, providing explicit value is **mandatory**. If object area is absent, commondata::missing value is used as a default, which results in **wrong calculations**.

- Such a case is logged with the commondata::ltag_error logger tag. (check logger errors with `grep ERROR: *log`).

Check optional `contrast` parameter. If unset, use global `commondata::preycontrast_default`.
Check optional time step parameter. If unset, use global `commondata::global_time_step_model_current`.
Calculate ambient illumination / irradiance at the depth of this object at the given time step.
Return visual range to see this spatial object: its visibility range.
Definition at line 1938 of file m_env.f90.

8.5.3.29 spatial_get_environment_in_pos()

```
pure integer function the_environment::spatial_get_environment_in_pos (
            class(spatial), intent(in) this,
            class(environment), dimension(:), intent(in), optional environments_array )
```

Identify in which environment from the input list this spatial agent is currently in. Example call:

```
ienv = object%find_environment( [habitat_safe, habitat_dangerous] )
```

Note

Because the habitat object is an extension of the environment, this method also works with the habitats.

Determining the environment object the agent is currently in can be done by the_environment::spatial::find_environment() method in this way:

```
...
environment_limits = Global_Habitats_Available(                        &
               this_agent%find_environment(Global_Habitats_Available) )
...
```

This uses the the_environment::global_habitats_available global array containing all available environments, initialised in the_evolution::init_environment_objects().

Parameters

in	*environments_array*	environments_array An array of environment objects, where the `this` object can be. If this parameter is omitted, the environment objects are obtained from the default global array the_environment::global_habitats_available.

Warning

The environment objects within the array must be non-overlapping, otherwise, the results are undefined due to parallel algorithm.

Returns

Returns the number of the environment from the input array, where this spatial object is currently in.

8.5.3.29.1 Implementation details First, determine the size of the input array of the environments among which the check is done.
Also, initialise the return value to zero.
Then cycle over all the input environments whether the this spatial object is within it.
The `.within.` operator is used for checking (defined by `the_environment::environment_check_located_within_`
Then, return the number of the environment within the input array and exit.
If the `environments_array` array is not provided, default habitats are obtained from the the_environment::global_habitats_availab
global array .
Definition at line 2039 of file m_env.f90.

8.5.3.30 spatial_moving_fix_position_3d_v()

```
subroutine the_environment::spatial_moving_fix_position_3d_v (
             class(spatial_moving), intent(inout) this,
             real(srp), intent(in) x,
             real(srp), intent(in) y,
             real(srp), intent(in) depth )
```

Place spatial movable object into a 3D space, define the object's current coordinates, but first save previous coordinates.

Parameters

x	The spatial objects will now get these coordinates.
y	The spatial objects will now get these coordinates.
depth	The spatial objects will now get these coordinates.

8.5.3.30.1 Implementation details Save previous coordinates into the history stacks.
Finally, position the object now with the coordinates provided.
Definition at line 2121 of file m_env.f90.

8.5.3.31 spatial_moving_fix_position_3d_o()

```
elemental subroutine the_environment::spatial_moving_fix_position_3d_o (
             class(spatial_moving), intent(inout) this,
             type(spatial), intent(in) location )
```

Place spatial movable object into a 3D space, define the object's current coordinates, but first save previous coordinates.

Parameters

location	The spatial objects will now get these coordinates.

8.5.3.31.1 Implementation details

Save previous coordinates into the history stacks.
Finally, position the object now with the coordinates provided.
Definition at line 2144 of file m_env.f90.

8.5.3.32 spatial_moving_repeat_position_history_3d()

```
elemental subroutine the_environment::spatial_moving_repeat_position_history_3d (
            class(spatial_moving), intent(inout) this )
```

Repeat (re-save) the current position into the positional history stack.

Note

Re-saving the current position is necessary to keep the full positional history even for the the_behavior↩ ::behaviour's that do not involve spatial displacement (movement).

Definition at line 2168 of file m_env.f90.

8.5.3.33 spatial_distance_3d()

```
elemental real(srp) function the_environment::spatial_distance_3d (
            class(spatial), intent(in) this,
            class(spatial), intent(in) other )
```

Calculate the Euclidean distance between two spatial objects. This is a type-bound function.

Returns

distance Euclidean distance between two spatial objects.

Parameters

other	another spatial object that we measure distance between.

Note

Note that this version uses the vector-based backend for calculation. The vector-based backend is equivalent to the scalar-based, but might be more general as it easily works with other dimensionality (e.g. 2D or 4D). The negative side is that the vector-based backend cannot be used to make an **elemental** function. **Example**↩ **:** `x_dist = objectdistance(other_object)`

8.5.3.33.1 Implementation details

Calculate the distance between these two spatial objects.

Note

Note that this version uses the vector-based backend for calculation. The vector-based backend is equivalent to the scalar-based, but might be more general as it easily works with other dimensionality (e.g. 2D or 4D). The negative side is that the vector-based backend cannot be used to make an **elemental** function.

Definition at line 2190 of file m_env.f90.

8.5.3.34 spatial_stack2arrays()

```
pure type(spatial) function, dimension(:), allocatable the_environment::spatial_stack2arrays (
```

```
            type(spatial), dimension(:), intent(in) a,
            type(spatial), dimension(:), intent(in) b )
```

Concatenate two arrays of the_environment::spatial objects `a` and `b`. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.

Note

There is a user defined operator `.cat.` making use of this procedure that can be used like this:
`object1%location() .cat. object2%location()`

Warning

This is the the_environment::spatial **type** version. All input and output parameters are defined as **type**, so this is not class-safe.

Parameters

in	*a*	a first array
in	*b*	b second array

Returns

an array [a, b]

Definition at line 2223 of file m_env.f90.

8.5.3.35 spatial_moving_stack2arrays()

```
pure type(spatial_moving) function, dimension(:), allocatable the_environment::spatial_↩
moving_stack2arrays (
            type(spatial_moving), dimension(:), intent(in) a,
            type(spatial_moving), dimension(:), intent(in) b )
```

Concatenate two arrays of the_environment::spatial_moving objects `a` and `b`. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.

Note

There is a user defined operator `.cat.` making use of this procedure that can be used like this:
`object1%location() .cat. object2%location()`

Warning

This is the the_environment::spatial_moving **type** version. All input and output parameters are defined as **type**, so this is not class-safe.

Parameters

in	*a*	a first array
in	*b*	b second array

Returns

an array [a, b]

Definition at line 2250 of file m_env.f90.

8.5.3.36 spatial_class_stack2arrays_locs()

```
pure type(spatial) function, dimension(:), allocatable the_environment::spatial_class_stack2arrays↩
_locs (
             class(spatial), dimension(:), intent(in) a,
             class(spatial), dimension(:), intent(in) b )
```

Concatenate the **location** components of two arrays of the_environment::spatial class objects a and b. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.

Note

There is a user defined operator `.cat.` making use of this procedure that can be used like this:
`all_objects%position%( object1 .catloc. object2 )`

Warning

Unlike the the_environment::spatial_stack2arrays() and the_environment::spatial_moving_stack2arrays() methods, this procedure is class-safe and can be used with any class upwards, but it concatenates **only** the location data (returns **type** the_environment::spatial).

Parameters

in	*a*	a first array
in	*b*	b second array

Returns

an array [a, b]

Definition at line 2280 of file m_env.f90.

8.5.3.37 dist3d()

```
elemental real(srp) function the_environment::dist3d (
             class(spatial), intent(in) this,
             class(spatial), intent(in) other )
```

This is a **non-type-bound** version of the distance calculation function.

Note

Note that this is an **elemental** function that can also accept arrays (but these must be conforming). Example:

```
x_dist = dist3d(object, other_object)    # scalar variant
d=dist3d( habitat_safe%food%food(1:3),                              &
          habitat_safe%food%food(4:6) )  # vector variant
```

Returns

distance Euclidean distance between two spatial objects.

Parameters

in	*other*	other another spatial object that we measure distance between.

8.5.3.37.1 Implementation details Calculate the distance between these two spatial objects.

Note

Note that this version uses the scalar-based backend for calculation. The scalar-based backend is equivalent to the vector-based one, but is linked with the 3D space only (has to be re-implemented in case of 2D or 4D space). But its positive side is that it is an **elemental** function that can be used to make further elemental functions.

Definition at line 2306 of file m_env.f90.

8.5.3.38 spatial_self_distance_3d()

```
elemental real(srp) function the_environment::spatial_self_distance_3d (
             class(spatial), intent(in) this,
             integer, intent(in), optional from_history )
```

Calculate the Euclidean distance between the current and previous position of a single spatial object.

Parameters

from_history	We have to calculate total distance from this point in the spatial stack history (< HISTORY_SIZE_SPATIAL). Not used here, always 0.0.

Returns

distance Euclidean distance between two spatial objects.

8.5.3.38.1 Implementation details The distance between two positions of an immobile object is zero in all cases. It is a fixed value in this procedure.
Definition at line 2334 of file m_env.f90.

8.5.3.39 spatial_moving_self_distance_3d()

```
elemental real(srp) function the_environment::spatial_moving_self_distance_3d (
             class(spatial_moving), intent(in) this,
             integer, intent(in), optional from_history )
```

Calculate the Euclidean distance between the current and previous position of a single spatial movable object. Optionally, it also calculates the total distance traversed during the `from_history` points from the history stack along with the distance from the current position and the last historical value. For example, to calculate the **average** distance throughout the whole history (`HISTORY_SIZE_SPATIAL` points) do this:

```
object_name%way(history_size_spatial - 1) / history_size_spatial
```

This is because for `N` points in history we can calculate `N-1` distances, but the sample size is `N`, that is `N-1` plus an additional distance between the latest historical point and the current position.

Parameters

from_history	We have to calculate total distance from this point in the spatial stack history (< `HISTORY_SIZE_SPATIAL`)

Returns

distance Euclidean distance between two spatial objects.

8.5.3.39.1 Implementation details We get the history size from the history stack size (`history`), alternatively, can get from the `HISTORY_SIZE_SPATIAL` parameter directly.
Check if we are asked to calculate distance traversed using the history stack. If the number provided exceed the limit, set maximum. If no history requested (parameter not present or 0) calculate the distance between the current point and the latest historical point.

Note

If we have history stack of size **N**, we can calculate maximum **N-1** historical distances between **N** successive points.

Calculate the distance between the current position and the last historical stack record.
Now cycle backwards through the `from_history` steps of the history stack and calculate the sum = total distance traversed.
Definition at line 2367 of file m_env.f90.

8.5.3.40 spatial_moving_create_3d()

```
elemental subroutine the_environment::spatial_moving_create_3d (
            class(spatial_moving), intent(inout) this )
```

Create a new spatial moving object. Initially it has no position, all coordinate values are `MISSING` or `INVALID` for real type coordinates.

Note

This is the **constructor** for `SPATIAL_MOVING` object type that makes it into existence and gives it its value.

8.5.3.40.1 Implementation details Assign `MISSING` values to the new object, use interface function and type constructor.
This also cleanups the history stack, i.e. fills it with `MISSING` values.
Definition at line 2435 of file m_env.f90.

8.5.3.41 spatial_moving_clean_hstory_3d()

```
elemental subroutine the_environment::spatial_moving_clean_hstory_3d (
            class(spatial_moving), intent(inout) this )
```

Create a new empty history of positions for spatial moving object. Assign all values to the MISSING value code.

8.5.3.41.1 Implementation details Set all historical stack **arrays** to `MISSING`.
Definition at line 2453 of file m_env.f90.

8.5.3.42 spatial_moving_go_up()

```
elemental subroutine the_environment::spatial_moving_go_up (
            class(spatial_moving), intent(inout) this,
            real(srp), intent(in), optional step )
```

The spatial moving object **ascends**, goes up the depth with specific fixed step size.

Parameters

`in`	*step*	step is the step size for the upwards walk. If it is too large (the object would extend beyond its current environment), it is adjusted autiomatically.

8.5.3.42.1 Implementation details Calculate the minimum depth in the environment currently occupied by the spatial object.

- Use a combination of the_environment::spatial::find_environment() and the_environment::environment::depth_min().

Check if the target depth is likely to go beyond the environment depth limits and reduce the upwnward walk step size accordingly. Namely, if the depth coordinate of the object minus the depth step exceeds the minimum depth, the step is reduced to be within the available environment: $d_o - D_{min} - \varepsilon$, where D_{min} is the maximum depth, d_o is the current depth of the object and ε is a very small constant defined by the parameter commondata::zero to guarantee the object remains within the current environment.

Relocate the object so that its X and Y coordinates remain intact but the depth reduces by the step size.

- Here, if the object is a the_environment::food_item class object, also check if it is the_environment::food_item::is_available() and move the object only if yes.

Definition at line 2467 of file m_env.f90.

8.5.3.43 spatial_moving_go_down()

```
elemental subroutine the_environment::spatial_moving_go_down (
            class(spatial_moving), intent(inout) this,
            real(srp), intent(in), optional step )
```

The spatial moving object **decends**, goes down the depth with specific fixed step size.

Parameters

in	*step*	step is the step size for the upwards walk. If it is too large (the object would extend beyond its current environment), it is adjusted autiomatically.

8.5.3.43.1 Implementation details Calculate the maximum depth in the environment currently occupied by the spatial object.

- Use a combination of the_environment::spatial::find_environment() and the_environment::environment::depth_max().

Check if the target depth is likely to go beyond the environment depth limits and reduce the downward walk step size accordingly. Namely, if the depth coordinate of the object plus the depth step exceeds the maximum depth, the step is reduced to be within the available environment: $D_{max} - d_o - \varepsilon$, where D_{max} is the maximum depth, d_o is the current depth of the object and ε is a very small constant defined by the parameter commondata::zero to guarantee the object remains within the current environment.
Relocate the object so that its X and Y coordinates remain intact but the depth reduces by the step size.

- Here, if the object is a the_environment::food_item class object, also check if it is the_environment::food_item::is_available() and move the object only if yes.

Definition at line 2530 of file m_env.f90.

8.5.3.44 spatial_moving_randomwalk_gaussian_step_3d()

```
subroutine the_environment::spatial_moving_randomwalk_gaussian_step_3d (
            class(spatial_moving), intent(inout) this,
            real(srp), intent(in) meanshift,
            real(srp), intent(in) cv_shift,
            class(environment), intent(in), optional environment_limits )
```

Implements an optionally environment-restricted Gaussian random walk in 3D.

Parameters

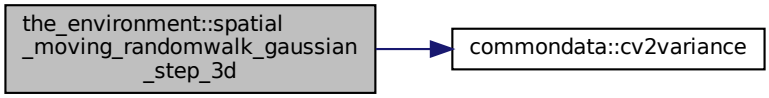
meanshift	the mean shift along any of the three dimensions.
cv_shift	the coefficient of variation for a single elementary shift in any of the three dimensions.
environment_limits	Limits of the environment area available for the random walk. The moving object cannot get beyond this limit.

The moving object walks in three dimensions. The process is simple, first shift along the x axis for some random Gaussian length (with the mean `meanshift` and the variance/CV `cv_shift`) then walk another Gaussian length the y axis. The, walk in the same manner along the z axis. The optional restriction is that the whole walk must not exceed specific spatial location set by the environment parameter.

8.5.3.44.1 Implementation details First, we get the current coordinates of the spatial object.
And set a temporary spatial test object with the new coordinates, advancing/adding our random walk step. This is done via the `.radd.` operator and calling RNORM (see HEDTOOLS).
Environment restriction part of the random walk, if `environment_limits` parameter object is provided. Here the object is not allowed to go beyond its bounding environment: get new position while outside of the target environment.

- Loop while this new test spatial object is outside of our target environment. It must be **strictly** within.

Finally, change the current position of the `this` object to the position defined by the `test_object`. The standard function `position` for the `SPATIAL_MOVING`is used, that keeps the movement history.
Definition at line 2604 of file m_env.f90.
Here is the call graph for this function:



8.5.3.45 spatial_moving_randomwalk_gaussian_step_25d()

```
subroutine the_environment::spatial_moving_randomwalk_gaussian_step_25d (
             class(spatial_moving), intent(inout) this,
             real(srp), intent(in) meanshift_xy,
             real(srp), intent(in) cv_shift_xy,
             real(srp), intent(in) meanshift_depth,
             real(srp), intent(in) cv_shift_depth,
             class(environment), intent(in), optional environment_limits )
```

Implements an optionally environment-restricted Gaussian random walk in a "2.5 dimensions", i.e. 2D x y with separate walk parameters for the third depth dimension.

Parameters

meanshift	the mean shift along any of the three dimensions.
cv_shift	the coefficient of variation for a single elementary shift in any of the three dimensions.
environment_limits	Limits of the environment area available for the random walk. The moving object cannot get beyond this limit.

The moving object walks in three dimensions. The process is simple, first shift along the x axis for some random Gaussian length (with the mean `meanshift` and the variance/CV `cv_shift`) then walk another Gaussian length the y axis. The, walk in the same manner along the z axis. But here z axis has separate walk parameters (`meanshift` and `cv_shift`) that are much smaller than the x and y parameters. The optional restriction is that the whole walk must not exceed specific spatial location set by the environment parameter.

Note

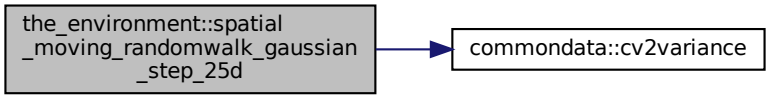
The mean and CV is different for the 2D x y movement and the third dimension *depth* movement.

8.5.3.45.1 Implementation details First, we get the current coordinates of the spatial object.
And set a temporary spatial test object with the new coordinates, advancing/adding our random walk step. This is done via the `.radd.` operator and calling the `RNORM` function (see HEDTOOLS).

Environment restriction part of the random walk, if `environment_limits` parameter object is provided. Here the object is not allowed to go beyond its bounding environment: get new position while outside of the target environment.

- Loop while this new test spatial object is outside of our target environment. It must be **strictly** within.

Finally, change the current position of the `this` object to the position defined by the `test_object`. The standard function `position` for the `SPATIAL_MOVING`is used, that keeps the movement history.
Definition at line 2686 of file m_env.f90.
Here is the call graph for this function:



8.5.3.46 spatial_moving_corwalk_gaussian_step_3d()

```
subroutine the_environment::spatial_moving_corwalk_gaussian_step_3d (
             class(spatial_moving), intent(inout) this,
             class(spatial), intent(in) target,
             real(srp), intent(in) meanshift,
             real(srp), intent(in) cv_shift,
             logical, intent(in), optional is_away,
             real(srp), intent(in), optional ci_lim,
             class(environment), intent(in), optional environment_limits,
             logical, intent(out), optional is_converged,
             integer, intent(out), optional debug_reps )
```

Implements an optionally environment-restricted **correlated directional** Gaussian random walk in 3D towards (or away of) an the_environment::spatial class `target` object.
The moving object walks in three dimensions towards (or away of) a the_environment::spatial class target. Here is an example of walks:

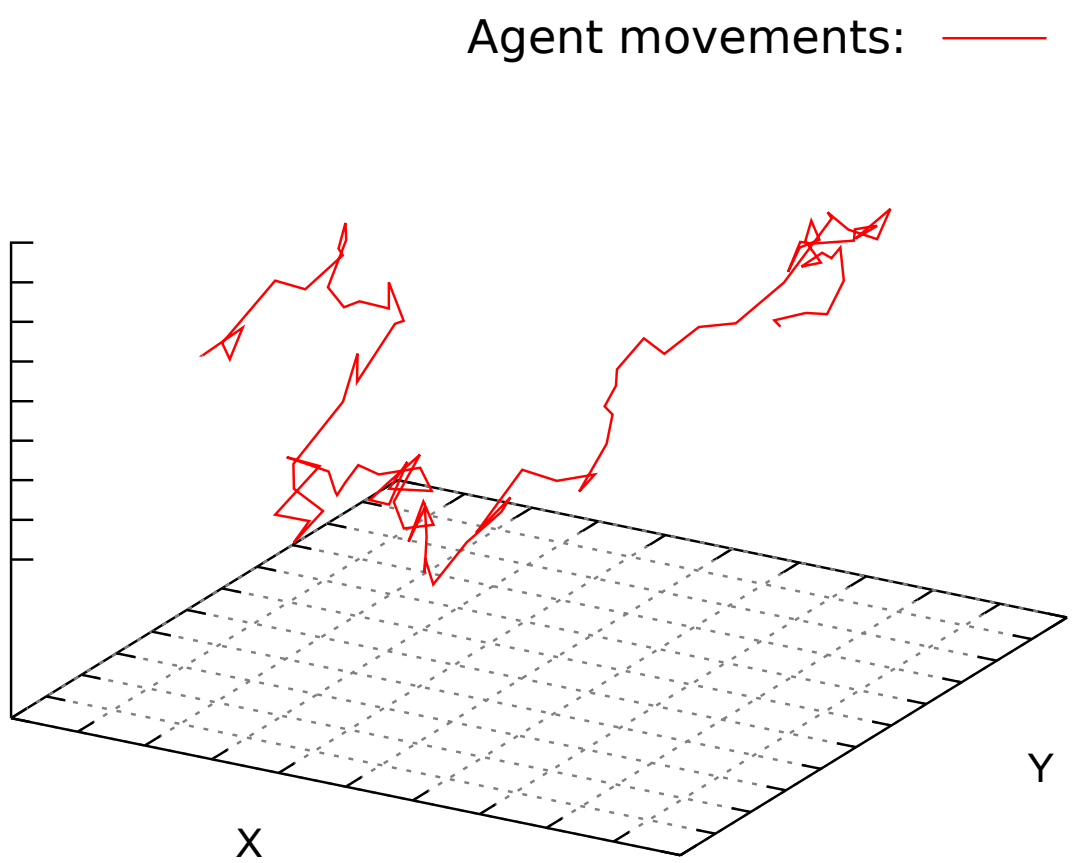
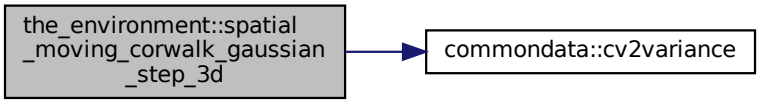


Figure 8.7 Correlated Gaussian random walk in 3D

Parameters

in	*target*	target The target of the random walk.
in	*meanshift*	meanshift the mean shift along any of the three dimensions.
in	*cv_shift*	cv_shift the coefficient of variation for a single elementary shift in any of the three dimensions.
in	*is_away*	is_away optional logical flag, if set to TRUE, the walk is actually directed out of the target, so that the object is going to maximise rather than minimise the distance to the target.
in	*ci_lim*	ci_lim This parameter sets the convergence criterion; it is the confidence interval limit for the distance between the object and the target. The walk is considered "converged" if the distance between the object and the target is smaller than `ci_lim` std. deviations of the average step distance (set by `meanshift`). The default value is the 95% confidence interval (1.95996).
in	*environment_limits*	environment_limits optional limits of the environment area available for the random walk. The moving object cannot get beyond this limit.
out	*is_converged*	is_converged optional logical flag for the "converged" state, i.e. the distance between the object and the target is smaller than `ci_lim` standard deviations of the average step distance.
out	*debug_reps*	debug_reps optional internal counter for repeated samplings of random positions for prospective spatial advancement of the object. If the counter reaches a too high value, a "no convergence" state is reached. In case of the avoiding environmentally limited walk, when the object maximises its distance from the target, such a condition may indicate that there is no further space to avoid the target in the available environment.

8.5.3.46.1 Implementation details First, check if the convergence criterion is reached. The walk is considered "converged" if the distance between the object and the target is smaller than `ci_lim` std. deviations of the average step distance (which is set by `meanshift`).
If the convergence condition is met, the object *does not* change its position any more, no further walks are performed.
If the convergence condition is not yet met, the current coordinates of the spatial object are recorded.
Then a temporary spatial test object (`test_object`) is set the new coordinates for the spatial moving object, advancing to a random walk step. This is done by randomly adding or subtracting a Gaussian step size with the mean `meanshift` and variance defined by `cv_shift` along all three spatial coordinates.
A series of conditions is then checked, the main is that the new position defined by the temporary spatial object should be such that the distance from the target must be smaller (if the object is intended to moved towards the target) or larger (if the object is intended to move away of the target).
Environment restriction is also applied if `environment_limits` parameter is provided: the object is not allowed to go beyond its bounding environment in such a case.
There is also a safeguard against poor convergence: If the number of iterations exceeds a fixed value (`CONVERG` local parameter), force to exit from the condition loops without changing the object position (i.e. no walk is done).
Finally, change the current position of the `this` object to the position defined by the `test_object`. Such a change is done only if the non-convergence condition is not detected.
At the end, check and return the optional intent[out] parameters `is_converged` and `debug_reps`.
Definition at line 2765 of file m_env.f90.
Here is the call graph for this function:



8.5.3.47 spatial_moving_corwalk_gaussian_step_25d()

```
subroutine the_environment::spatial_moving_corwalk_gaussian_step_25d (
             class(spatial_moving), intent(inout) this,
             class(spatial), intent(in) target,
             real(srp), intent(in) meanshift_xy,
             real(srp), intent(in) cv_shift_xy,
             real(srp), intent(in) meanshift_depth,
             real(srp), intent(in) cv_shift_depth,
             logical, intent(in), optional is_away,
             real(srp), intent(in), optional ci_lim,
             class(environment), intent(in), optional environment_limits,
             logical, intent(out), optional is_converged,
             integer, intent(out), optional debug_reps )
```

Implements an optionally environment-restricted **correlated directional** Gaussian random walk in 3D towards (or away of) an the_environment::spatial class `target` object.
The moving object walks in three dimensions towards (or away of) a the_environment::spatial class target.

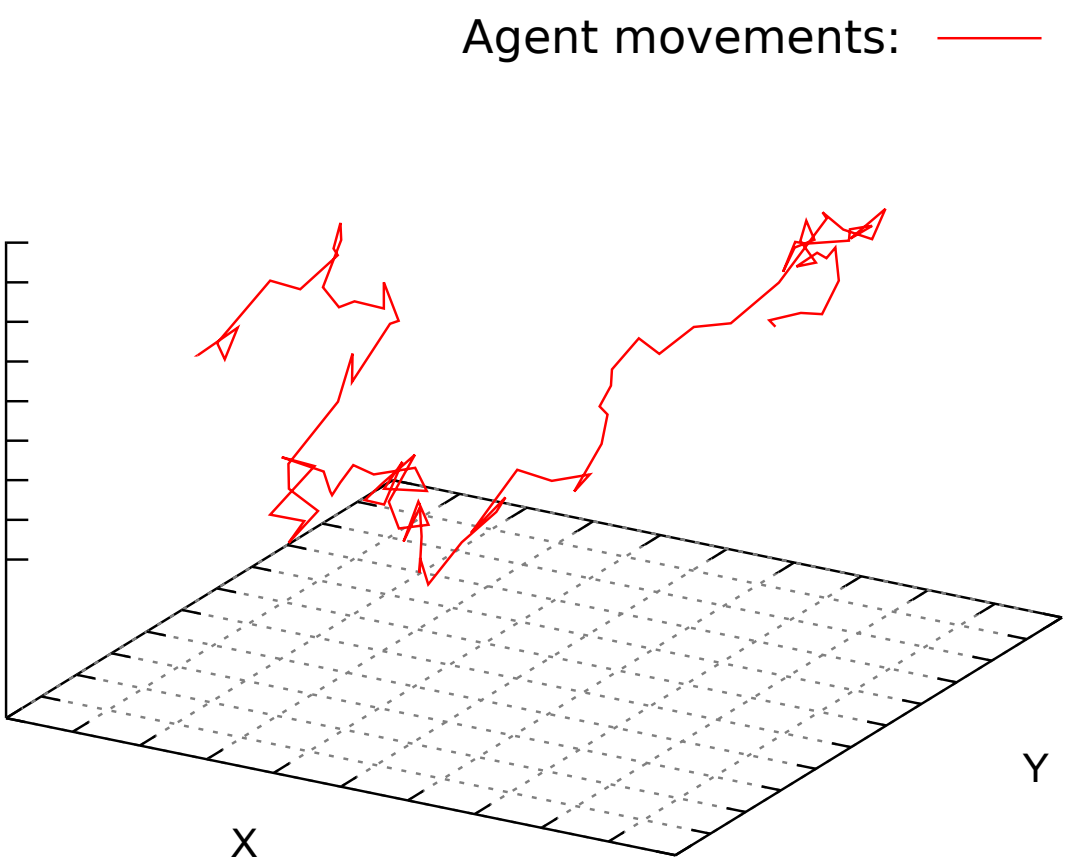


Figure 8.8 Correlated Gaussian random walk in 3D

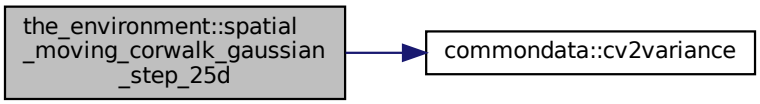
Parameters

in	*target*	target The target of the random walk.
in	*meanshift_xy*	meanshift_xy the mean shift along any of the three dimensions.
in	*cv_shift_xy*	cv_shift_xy the coefficient of variation for a single elementary shift in any of the three dimensions.
in	*meanshift_depth*	meanshift_depth the mean shift along any of the three dimensions.
in	*cv_shift_depth*	cv_shift_depth the coefficient of variation for a single elementary shift in any of the three dimensions.
in	*is_away*	is_away optional logical flag, if set to TRUE, the walk is actually directed out of the target, so that the object is going to maximise rather than minimise the distance to the target.
in	*ci_lim*	ci_lim This parameter sets the convergence criterion; it is the confidence interval limit for the distance between the object and the target. The walk is considered "converged" if the distance between the object and the target is smaller than `ci_lim` std. deviations of the average step distance (set by `meanshift`). The default value is the 95% confidence interval (1.95996).
in	*environment_limits*	environment_limits optional limits of the environment area available for the random walk. The moving object cannot get beyond this limit.
out	*is_converged*	is_converged optional logical flag for the "converged" state, i.e. the distance between the object and the target is smaller than `ci_lim` standard deviations of the average step distance.

Parameters

out	*debug_reps*	debug_reps optional internal counter for repeated samplings of random positions for prospective spatial advancement of the object. If the counter reaches a too high value, a "no convergence" state is reached. In case of the avoiding environmentally limited walk, when the object maximises its distance from the target, such a condition may indicate that there is no further space to avoid the target in the available environment.

8.5.3.47.1 Implementation details First, check if the convergence criterion is reached. The walk is considered "converged" if the distance between the object and the target is smaller than `ci_lim` std. deviations of the average step distance (which is set by `meanshift`).
If the convergence condition is met, the object *does not* change its position any more, no further walks are performed.
If the convergence condition is not yet met, the current coordinates of the spatial object are recorded.
Then a temporary spatial test object (`test_object`) is set the new coordinates for the spatial moving object, advancing to a random walk step. This is done by randomly adding or subtracting a Gaussian step size with the mean `meanshift` and variance defined by `cv_shift` along all three spatial coordinates.
A series of conditions is then checked, the main is that the new position defined by the temporary spatial object should be such that the distance from the target must be smaller (if the object is intended to moved towards the target) or larger (if the object is intended to move away of the target).
Environment restriction is also applied if `environment_limits` parameter is provided: the object is not allowed to go beyond its bounding environment in such a case.
There is also a safeguard against poor convergence: If the number of iterations exceeds a fixed value (`CONVERG` local parameter), force to exit from the condition loops without changing the object position (i.e. no walk is done).
Finally, change the current position of the `this` object to the position defined by the `test_object`. Such a change is done only if the non-convergence condition is not detected.
At the end, check and return the optional intent[out] parameters `is_converged` and `debug_reps`.
Definition at line 3007 of file m_env.f90.
Here is the call graph for this function:



8.5.3.48 spatial_moving_dirwalk_gaussian_step_3d()

```
subroutine the_environment::spatial_moving_dirwalk_gaussian_step_3d (
             class(spatial_moving), intent(inout) this,
             class(spatial), intent(in) target,
             real(srp), intent(in) meanshift,
             real(srp), intent(in) cv_shift,
             class(environment), intent(in), optional environment_limits )
```

Implements an optionally environment-restricted **directional** Gaussian random walk in 3D towards a `target` the_environment::spatial class object.
The moving object walks in three dimensions towards a target. The process is simple, first shift along the x axis for some random Gaussian length (with the mean `meanshift` and the variance/CV `cv_shift`) in the direction that minimises the coordinate-bound distance from the target. If the target is located at a distance not exceeding

the meanshift we have got towards the target. Then the process is repeated for the y and z axes. The optional restriction is that the whole walk must not exceed specific spatial location set by the environment parameter.

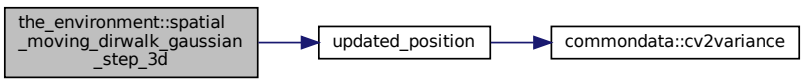
Note

This dirwalk is a obsolete suboptimal implementation See the_environment::spatial_moving_corwalk_gaus and the_environment::spatial_moving_corwalk_gaussian_step_25d() for a better alternative.

Parameters

in	*target*	The target of the random walk, the walk should converge to the target within finite number of steps.
in	*meanshift*	the mean shift along any of the three dimensions.
in	*cv_shift*	the coefficient of variation for a single elementary shift in any of the three dimensions.
in	*environment_limits*	Limits of the environment area available for the random walk. The moving object cannot get beyond this limit.

8.5.3.48.1 Implementation details First, we get the current coordinates of the spatial object.
And set a temporary spatial test object with the new coordinates, advancing/adding our random walk step. This is done via the updated_position sub-function.
Environment restriction part of the random walk, if environment_limits parameter object is provided. Here the object is not allowed to go beyond its bounding environment.
Loop while this new test spatial object is outside our target environment. It must be **strictly** within.
(if the test_object is outside the environment, we create new randomly updated coordinates.)
Finally, change the current position of the this object to the position defined by the test_object. The standard function position for the SPATIAL_MOVINGis used, that keeps the movement history.
Definition at line 3288 of file m_env.f90.
Here is the call graph for this function:



8.5.3.49 spatial_moving_dirwalk_gaussian_step_25d()

```
subroutine the_environment::spatial_moving_dirwalk_gaussian_step_25d (
             class(spatial_moving), intent(inout) this,
             class(spatial), intent(in) target,
             real(srp), intent(in) meanshift_xy,
             real(srp), intent(in) cv_shift_xy,
             real(srp), intent(in) meanshift_depth,
             real(srp), intent(in) cv_shift_depth,
             class(environment), intent(in), optional environment_limits )
```

Implements an optionally environment-restricted **directional** Gaussian random walk in "2.5"-D towards a target the_environment::spatial class object. i.e. 2D x y with separate walk parameters for the third depth dimension.
The moving object walks in three dimensions towards a target. The process is simple, first shift along the x axis for some random Gaussian length (with the mean meanshift and the variance/CV cv_shift) in the direction that minimises the coordinate-bound distance from the target. If the target is located at a distance not exceeding

the meanshift we have got towards the target. Then the process is repeated for the y and z axes. The optional restriction is that the whole walk must not exceed specific spatial location set by the environment parameter.

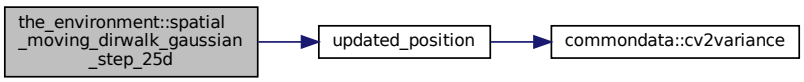
Note

This dirwalk is a obsolete suboptimal implementation See the_environment::spatial_moving_corwalk_gaus and the_environment::spatial_moving_corwalk_gaussian_step_25d() for a better alternative.

Parameters

in	*target*	The target of the random walk, the walk should converge to the target within finite number of steps.
in	*meanshift_xy*	the mean shift along the X and Y dimensions.
in	*cv_shift_xy*	the coefficient of variation for a single elementary shift the X and Ydimensions.
in	*meanshift_depth*	the mean shift along the depth dimension.
in	*cv_shift_depth*	the coefficient of variation for a single elementary shift in the depth dimension.
in	*environment_limits*	Limits of the environment area available for the random walk. The moving object cannot get beyond this limit.

8.5.3.49.1 Implementation details First, we get the current coordinates of the spatial object.
And set a temporary spatial test object with the new coordinates, advancing/adding our random walk step. This is done via the updated_position sub-function.
Environment restriction part of the random walk, if environment_limits parameter object is provided. Here the object is not allowed to go beyond its bounding environment.
Loop while this new test spatial object is outside our target environment. It must be **strictly** within.
(if the test_object is outside the environment, we create new randomly updated coordinates.)
Finally, change the current position of the this object to the position defined by the test_object. The standard function position for the SPATIAL_MOVINGis used, that keeps the movement history.
Definition at line 3410 of file m_env.f90.
Here is the call graph for this function:



8.5.3.50 rwalk3d_array()

```
subroutine the_environment::rwalk3d_array (
            class(spatial_moving), dimension(:), intent(inout) this,
            real(srp), dimension(:), intent(in), optional dist_array,
            real(srp), dimension(:), intent(in), optional cv_array,
            real(srp), intent(in), optional dist_all,
            real(srp), intent(in), optional cv_all,
            class(environment), intent(in), optional environment_limits,
            integer, intent(in), optional n_walks )
```

Perform one or several steps of random walk by an array of the_environment::spatial_moving class objects. This is a 3D version with the same walk parameters for the horizontal *XxY* plane and *depth*.

Parameters

in,out	*this*	[in] this is an array of the_environment::spatial class objects.
in	*dist_array*	step_size_array an array of step sizes for each object.
in	*cv_array*	cv_array Coefficients of variation for the walk.
in	*dist_all*	dist_all the value of the walk step size that is identical in all objects in the array.
in	*cv_all*	cv_all the value of the walk coefficient of variation that is identical in all objects in the array.
in	*environment_limits*	environment_limits Limits of the environment area available for the random walk. The moving object cannot get beyond this limit. If this parameter is not provided, the environmental limits are obtained automatically from the global array the_environment::global_habitats_available.
in	*n_walks*	n_walk optional number of walk steps that should be performed, default just one.

8.5.3.50.1 Implementation details

- Calculate the distance array size.

calculate the step size along the axes from the distance array.

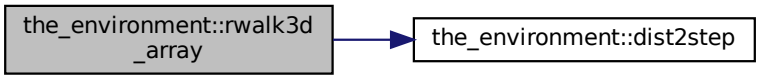
- Calculate the random permutation of individual indices.

  Warning

  Random order here is a prototype for testing for use in behaviour selection by population members.

- Perform Gaussian random walks for each of the individuals in a random order that is set by the `pop_↩ permutation` array.
- if the input objects array is of the_environment::food_item, also check if it is available (not eaten) using the the_environment::food_item::is_available() method.
- in the default class case, no such check is made.

Definition at line 3522 of file m_env.f90.
Here is the call graph for this function:



8.5.3.51 rwalk25d_array()

```
subroutine the_environment::rwalk25d_array (
            class(spatial_moving), dimension(:), intent(inout) this,
            real(srp), dimension(:), intent(in), optional dist_array_xy,
            real(srp), dimension(:), intent(in), optional cv_array_xy,
            real(srp), dimension(:), intent(in), optional dist_array_depth,
            real(srp), dimension(:), intent(in), optional cv_array_depth,
```

```
real(srp), intent(in), optional dist_all_xy,
real(srp), intent(in), optional cv_all_xy,
real(srp), intent(in), optional dist_all_depth,
real(srp), intent(in), optional cv_all_depth,
class(environment), intent(in), optional environment_limits,
integer, intent(in), optional n_walks )
```

Perform one or several steps of random walk by an array of the_environment::spatial_moving class objects. This is a 2.5D version with separate walk parameters for the horizontal *XxY* plane and *depth*.

Parameters

in,out	*this*	[in] this is an array of the_environment::spatial class objects.
in	*dist_array_xy*	dist_array_xy an array of step sizes for each object.
in	*cv_array_xy*	cv_array_xy Coefficients of variation for the walk.
in	*dist_array_depth*	dist_array_depth an array of step sizes for each object.
in	*cv_array_depth*	cv_array_depth Coefficients of variation for the walk.
in	*dist_all_xy*	dist_all_xy the value of the walk step size for horizontal plane that is identical in all objects in the array.
in	*cv_all_xy*	cv_all_xy the value of the walk coefficient of variation in the horizontal plane that is identical in all objects within the array.
in	*dist_all_depth*	dist_all_depth the value of the walk step size for the depth plane that is identical in all objects in the array.
in	*cv_all_depth*	cv_all_depth the value of the walk coefficient of variation in the depth plane that is identical in all objects in the array.
in	*environment_limits*	environment_limits Limits of the environment area available for the random walk. The moving object cannot get beyond this limit. If this parameter is not provided, the environmental limits are obtained automatically from the global array the_environment::global_habitats_available.
in	*n_walks*	n_walk optional number of walk steps that should be performed, default just one.

8.5.3.51.1 Implementation details

- Calculate the distance array size.

If the depth walk step distance is not provided as a parameter, 1/2 of the default step size is used as the default value. Thus, it is assumed that the extent of random movements of the agents in the horizontal plane is greater than vertical movements.

- Calculate the step size along the axes from the distance array.
- Calculate the random permutation of individual indices.

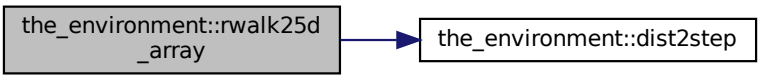
  Warning

  Random order here is a prototype for testing for use in behaviour selection by population members.

- Perform Gaussian random walks for each of the objects in a random order that is set by the pop_↩ permutation array.
- if the input objects array is of the_environment::food_item, also check if it is available (not eaten) using the the_environment::food_item::is_available() method.
- in the default class case, no such check is made.

Definition at line 3672 of file m_env.f90.

Here is the call graph for this function:



8.5.3.52 spatial_check_located_within_3d()

```
elemental logical function the_environment::spatial_check_located_within_3d (
            class(spatial), intent(in) this,
            class(environment), intent(in) environment_limits )
```

Function to check if this spatial object is located within an area set by an environmental object (parameter). This should be similar to an analogous function defined for the environment object.

Parameters

environment_limits	`the_environment::environment` object that sets the limits that we check the current spatial object to be within.

Returns

Logical flag TRUE if the the_environment::spatial object is within the `environment_limits` environment.

Note

Can be used as a user-defined operator:
`if ( object .within. environment ) then`

We need it to implement environment-restricted Gaussian random walk.

Definition at line 3891 of file m_env.f90.

8.5.3.53 spatial_check_located_below()

```
elemental logical function the_environment::spatial_check_located_below (
            class(spatial), intent(in) this,
            class(spatial), intent(in) check_object )
```

Logical function to check if the **argument** spatial object(s) (`check_object`) is (are) located **below** the **this** reference spatial object. Elemental function that also works with arrays. Use as:
`reference_object%is_below(check_object)`
See also the user-defined operator `.below.`. The `.below.` operator can be used in two ways:

- as an expression, with both scalar and array values:
  `parents%ind(i) .below. parents%ind(i)%perceive_food%foods_seen`
- in if blocks, only **scalars**:
  `if ( parents%ind(i) .below. parents%ind(i)%perceive_food%foods_seen(1) )`

Definition at line 3924 of file m_env.f90.

8.5.3.54 spatial_check_located_above()

```
elemental logical function the_environment::spatial_check_located_above (
             class(spatial), intent(in) this,
             class(spatial), intent(in) check_object )
```

Logical function to check if the **argument** spatial object(s) (`check_object`) is (are) located **above** the **this** reference spatial object. Elemental function that also works with arrays. Use as:
`reference_object%is_above(check_object)`
See also the user-defined operator `.above.`. The `.above.` operator can be used in two ways:

- as an expression, with both scalar and array values:
  `parents%ind(i) .above. parents%ind(i)%perceive_food%foods_seen`
- in if blocks, only **scalars**:
  `if ( parents%ind(i) .above. parents%ind(i)%perceive_food%foods_seen(1) )`

Definition at line 3956 of file m_env.f90.

8.5.3.55 spatial_get_nearest_object()

```
type(spatial) function the_environment::spatial_get_nearest_object (
             class(spatial), intent(in) this,
             class(spatial), dimension(:), intent(in) neighbours,
             integer, intent(out), optional number )
```

Determine the nearest spatial object to **this** spatial object among an array of other spatial objects.

Note

These two functions closely related, they return the nearest *object* and its *id*

- the_environment::spatial_get_nearest_object()
- the_environment::spatial_get_nearest_id()

However, each of them can also return the other output parameter as an intent(out) optional argument (in the subroutine style). For example the_environment::spatial_get_nearest_object() returns the nearest *object* but can also provide its *id* as an intent(out) dummy parameter. This is done for convenience of the function use.

Parameters

neighbours	the array of spatial objects that we search for the nearest one.
number	Optional number of the nearest object within the neighbours array.

Returns

Returns the nearest spatial object among the array.

Note

This function returns the nearest spatial **object itself** with optionally its number. See also the next function `spatial_get_nearest_id`.

8.5.3.55.1 Implementation details Calculate an array of distances between this object and the neighbouring objects using the_environment::spatial::distance().
Locate the index of the minimum distance.
And return the the_environment::spatial::location() function result for this neighbour as the **resultant object**.
Also return the optional nearest neighbour index number if requested.
Definition at line 3992 of file m_env.f90.

8.5.3.56 spatial_get_nearest_id()

```
integer function the_environment::spatial_get_nearest_id (
```

```
            class(spatial), intent(in) this,
            class(spatial), dimension(:), intent(in) neighbours,
            type(spatial), intent(out), optional object )
```

Determine the nearest spatial object to **this** spatial object among an array of other spatial objects.

Note

These two functions closely related, they return the nearest *object* and its *id*

- the_environment::spatial_get_nearest_object()
- the_environment::spatial_get_nearest_id()

However, each of them can also return the other output parameter as an intent(out) optional argument (in the subroutine style). For example the_environment::spatial_get_nearest_object() returns the nearest *object* but can also provide its *id* as an intent(out) dummy parameter. This is done for convenience of the function use.

Parameters

neighbours	the array of spatial objects that we search for the nearest one.
object	optional spatial object that is the nearest neighbour to **this** object.

Returns

The id number of the nearest object within the neighbours array

Note

This function returns the **id number** of the nearest spatial object and optionally the object itself. See also the previous function `spatial_get_nearest_object`.

8.5.3.56.1 Implementation details Calculate an array of distances between this object and the neighbouring objects. Note that we cannot use a whole-array single-liner as `distance` is a type bound *function*.
Return the index of the nearest neighbour.
Also return the optional nearest neighbour object itself if requested.
Definition at line 4063 of file m_env.f90.

8.5.3.57 habitat_make_init()

```
subroutine the_environment::habitat_make_init (
            class(habitat), intent(inout) this,
            type(spatial), intent(in) coord_min,
            type(spatial), intent(in) coord_max,
            character (len=*), intent(in), optional label,
            real(srp), intent(in), optional otherrisks,
            real(srp), intent(in), optional eggmortality,
            integer predators_number,
            type(spatial), dimension(:), optional loc_predators,
            integer food_abundance,
            type(spatial), dimension(:), optional loc_food,
            real(srp), dimension(:), optional sizes_food )
```

Make an instance of the habitat object (an environment superset).
Make / build habitat object and set parameters or defaults.

Warning

This subroutine seems to become quite *long* and difficult to understand. It also combines *several tasks*, making habitat limits, then array of predators then food resource. On the other hand, this init procedure is normally called only once. TODO: Consider splitting to a few shorter task-specific pieces.

8.5.3.57.1 Implementation details

8.5.3.57.1.1 A. Build the general properties of the habitat Build the physical spatial **environment** for this **habitat**, set the habitat coordinate limits.
Set label if provided or default random label otherwise.

8.5.3.57.1.2 B. Build the population of predators in the habitat Set the number of predators.
Allocate the local array of predator locations.
If we are provided with the array of spatial locations of each of the predators, we set the local array `loc_pred_here` from it.
If the array of locations is *not* present, construct *uniform* random distriibution of the predators locations within the present environment (bounded uniformly distributed locations). Now use the type bound function `uniform`.
Now we can **allocate** the **array of predators** in the habitat.

Note

Note that we have to allocate predators here, unlike the food resource part below as predators are just a raw array in this type definition. In contrast, food resource is an object itself with its own `make` procedure (that does allocation).

Make each of the predators using the call parameters of the object bound init function `make (attack_rate, position, label)`.

Note

Note that the attack rate is assumed to be Gaussian among the predators. Also, we do two separate loops for speed (avoid multiple repeated *if*'s within cycles)

If we happen to get zero variance, do deterministic predators.
Generate Gaussian stochastic predators with the object bound `make` procedure.
Deallocate the local array of predator locations, we do not need them any more further.

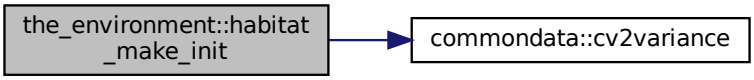
8.5.3.57.1.3 C. Build the food resource(s) of the habitat Set the number of food items in the food resource within the habitat.
Allocate the local array of food item locations.
If the array of locations is *not* present, construct *uniform* random distribution of the predators locations within the present environment (bounded uniformly distributed locations). Now use the type bound function `uniform`.
And also allocate the local array of food sizes
If the food size array `sizes_food` is provided, use it, if not, make a Gaussian stochastic food with parameters from `COMMONDATA`.
if we happened to get zero variance, do deterministic food resource.
Otherwise, generate random Gaussian array of food sizes.
Make the food resource now using the standard object-bound `make` function.

Note

Note that the food resource label is composed of `FOOD_` and the remaining part of the habitat label.

Note that we do **not** allocate the **food resource object** in this procedure as it is allocated automatically by the food-resource-bound subroutine `make`.

Deallocate temporary array at the end.
Definition at line 4116 of file m_env.f90.
Here is the call graph for this function:



8.5.3.58 habitat_name_get()

```
character(len=label_length) function the_environment::habitat_name_get (
            class(habitat), intent(in) this )
```

Return the name of the habitat.

Returns

The habitat name (text string label).

Definition at line 4342 of file m_env.f90.

8.5.3.59 habitat_get_risk_mortality()

```
real(srp) function the_environment::habitat_get_risk_mortality (
            class(habitat), intent(in) this )
```

Get the mortality risk associated with this habitat.

Returns

The mortality risk in this population that is not linked with explicit predation.

Definition at line 4353 of file m_env.f90.

8.5.3.60 habitat_get_risk_mortality_egg()

```
real(srp) function the_environment::habitat_get_risk_mortality_egg (
            class(habitat), intent(in) this )
```

Get the egg mortality risk associated with this habitat.

Returns

The mortality risk in this population that is not linked with explicit predation.

Definition at line 4365 of file m_env.f90.

8.5.3.61 habitat_save_predators_csv()

```
subroutine the_environment::habitat_save_predators_csv (
             class(habitat), intent(inout) this,
             character(len=*), intent(in), optional csv_file_name,
             logical, intent(out), optional is_success )
```

Save the predators with their characteristics into a CSV file.

Parameters

in	*csv_file_name*	csv_file_name optional file name to save the predators. Generated automatically if not provided.

8.5.3.61.1 Implementation notes First, check if the optional CSV file name is provided, if not, generate it automatically.
Second, save the predators data using the CSV_MATRIX_WRITE() from HEDTOOLS.
The CSV output data file can be optionally compressed with the commondata::cmd_zip_output command if commondata::is_zip_outputs is set to TRUE.
Definition at line 4377 of file m_env.f90.
Here is the call graph for this function:

8.5.3.62 save_dynamics()

```
subroutine the_environment::save_dynamics (
             real(srp), intent(in), optional maxdepth,
             character(len=*), intent(in) csv_file_name,
             logical, intent(out), optional is_success )
```

Save diagnostics data that shows the dynamics of the light and the average depth of the food items, light at the average depth of the food items etc at each time step of the model.
Code to generate plots from these data with gnuplot. Only 1 to 100s rows are plotted, pattern is sinusoidal.

```
set datafile separator ","
set xlabel "Time steps of the model"
set ylabel "SURFACE_LIGHT"
plot "init_dynamics.csv" every ::1::100 using 1:2 with lines, \
     "init_dynamics.csv" every ::1::100 using 1:3 with lines, \
     "init_dynamics.csv" every ::1::100 using 1:4 with lines, \
     "init_dynamics.csv" every ::1::100 using 1:5 with lines
```

Parameters

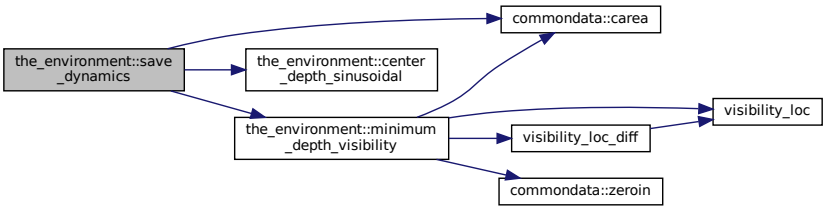
in	*maxdepth*	maxdepth is an optional maximum depth, if absent, is set to the global maximum depth (autodetected) across all habitats.
in	*csv_file_name*	csv_file_name is the file name to save the data. The file format is CSV.
out	*is_success*	is_success Flag showing that data save was successful (if TRUE).

The following data are saved in the CSV file:

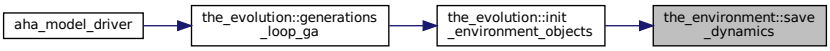
1. TIMESTEP – the time step of the model
2. SURFACE_LIGHT – light at the surface

3. LIGHT_DEP_10 – light at 1/10 of the maximum depth
4. LIGHT_DEP_HLF – light at 1/2 of the maximum depth
5. LIGHT_DEP_MAX – light at the maximum depth
6. MEAN_DEPTH – the mean depth of the food items (target depth)
7. LIGHT_MDEPTH – light at the mean depth of the food items.
8. FOOD_VIS_SURF – visibility range of a standard food item at the surface (zero depth)
9. FOOD_VIS_10 – visibility range of a standard food item at 1/10 of maximum depth
10. FOOD_VIS_HLF – visibility range of a standard food item at a half of the maximum depth.
11. FOOD_VIS_DPMAX – visibility range of a standard food item at the maximum depth
12. FOOD_VIS_MDEPT – visibility range of a standard food item at the target mean depth MEAN_DEPTH
13. DEP_VR_UND_200 – depth at which the visibility of the standard average food item falls below 100 cm
14. DEP_VR_UND_100 – depth at which the visibility of the standard average food item falls below 100 cm
15. DEP_VR_UND_020 – depth at which the visibility of the standard average food item falls below 20 cm
16. DEP_VR_UND_005 – depth at which the visibility of the standard average food item falls below 5 cm

Definition at line 4460 of file m_env.f90.
Here is the call graph for this function:



Here is the caller graph for this function:



8.5.3.63 environment_centre_coordinates_3d()

```
type(spatial) function the_environment::environment_centre_coordinates_3d (
             class(environment), intent(in) this,
             logical, intent(in), optional nodepth )
```

Determine the centroid of the environment.

Returns

habitat centre coordinates, spatial object type

Parameters

nodepth	Logical flag indicating that **depth** should not change.

Definition at line 4677 of file m_env.f90.

8.5.3.64 visual_range_scalar()

```
real(srp) function, private the_environment::visual_range_scalar (
            real(srp), intent(in) irradiance,
            real(srp), intent(in), optional prey_area,
            real(srp), intent(in), optional prey_contrast )  [private]
```

Wrapper for calculating *visual range of a fish predator* using the Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`. See `srgetr()` for computational details.

Note

Note that this is a **scalar** version. The measurement unit here is meter, might need conversion if other units are used.

Parameters

in	*irradiance*	background irradiance at specific depth
in	*prey_area*	prey area, m^2
in	*prey_contrast*	optional prey inherent contrast or default parameter if not present.

Returns

Returns visual range of the fish predator.

Example call:
```
visual_range( light_depth( 30., light_surface(100,.true.) ) )
```

8.5.3.64.1 Specific implementations

See specific implementations:

- the_environment::visual_range_scalar() for scalar argument
- the_environment::visual_range_vector() for vector argument
- the_environment::visual_range_fast() elemental (parallel-safe) version lacking sanity checks and extended debugging.

8.5.3.64.2 Notable parameters

8.5.3.64.2.1 VISUAL_RANGE_MAX_OVERFLOW

`VISUAL_RANGE_MAX_OVERFLOW = 1300.0_HRP`

The maximum ceiling value of visual range (cm) calculable using `srgetr()` under the `commondata::q_prec_128` numerical precision model (referred as commondata::hrp). This value is set to the visual range in the cases of numerical overflow.

Note

The real 128 bit limit (commondata::q_prec_128) is sufficient to calculate visual range up to the fish length of approximately 700 cm (area 153.9 cm^2). At this level, the maximum visual range is 1343.34 cm. This would be sufficient for non-whales.

8.5.3.64.2.2 error_msg

Local error message, character array:

- 1 = NO_CONVERGENCE
- 2 = DIVISION_ZERO
- 3 = NEGATIVE_RANGE

8.5.3.64.3 Implementation notes The computational backend for the visual range computation `srgetr()`, `easyr()` and `deriv()` now uses the `commondata::q_prec_128` 128 bit numerical precision model. This precision is sufficient to calculate visual range up to the the object radius of approximately 700 cm (area 153.9 cm^2) without x86 FPU overflow errors. At this level, the maximum visual range is 1343.34 cm.

8.5.3.64.3.1 Visual range plots The visual range plots below are generated by `HEDTOOLS/tools/visrange_plot.f90`.

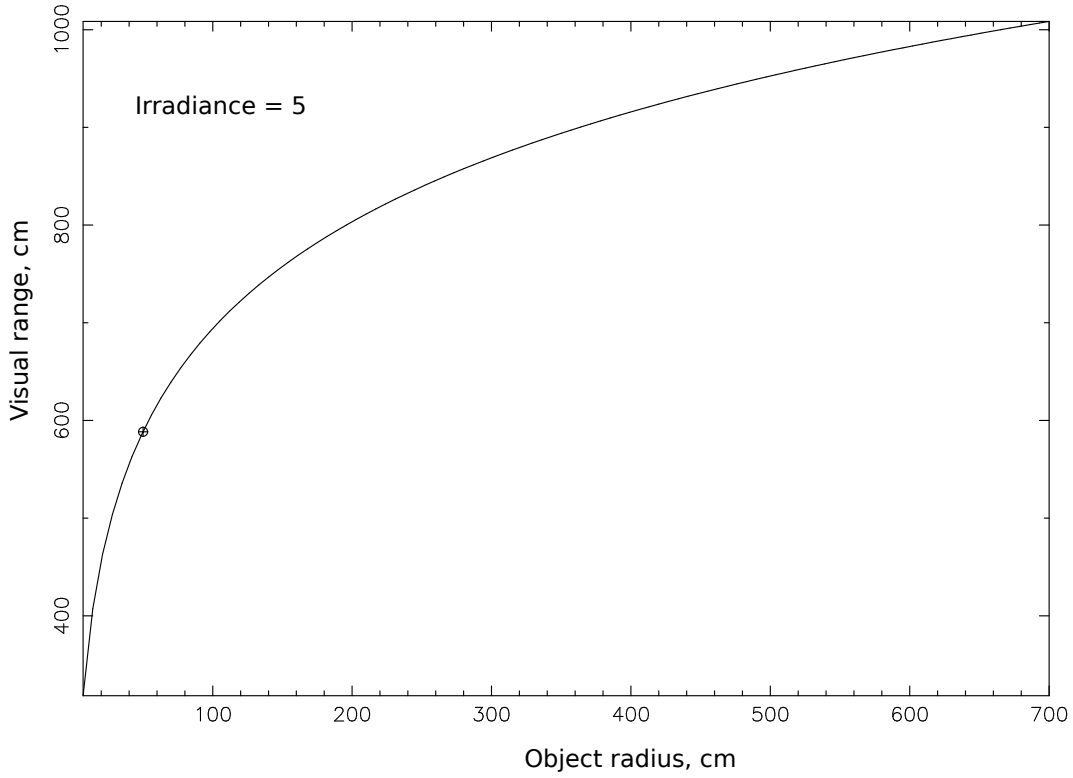


Figure 8.9 Visual range, 5.0

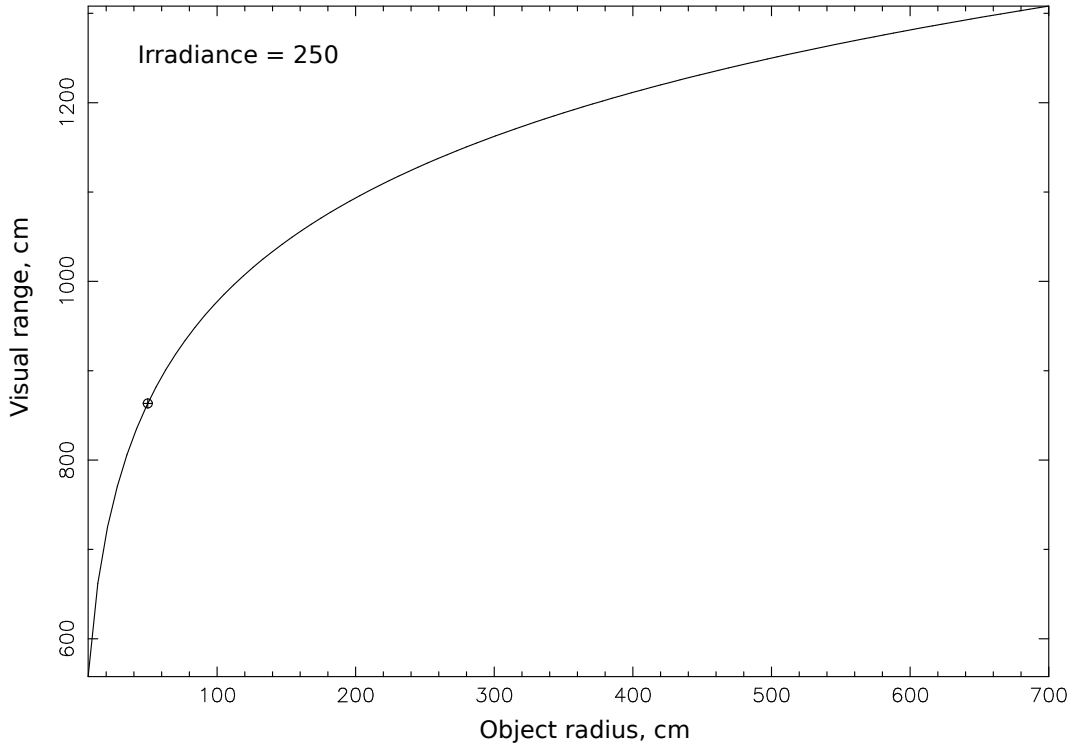


Figure 8.10 Visual range, 250

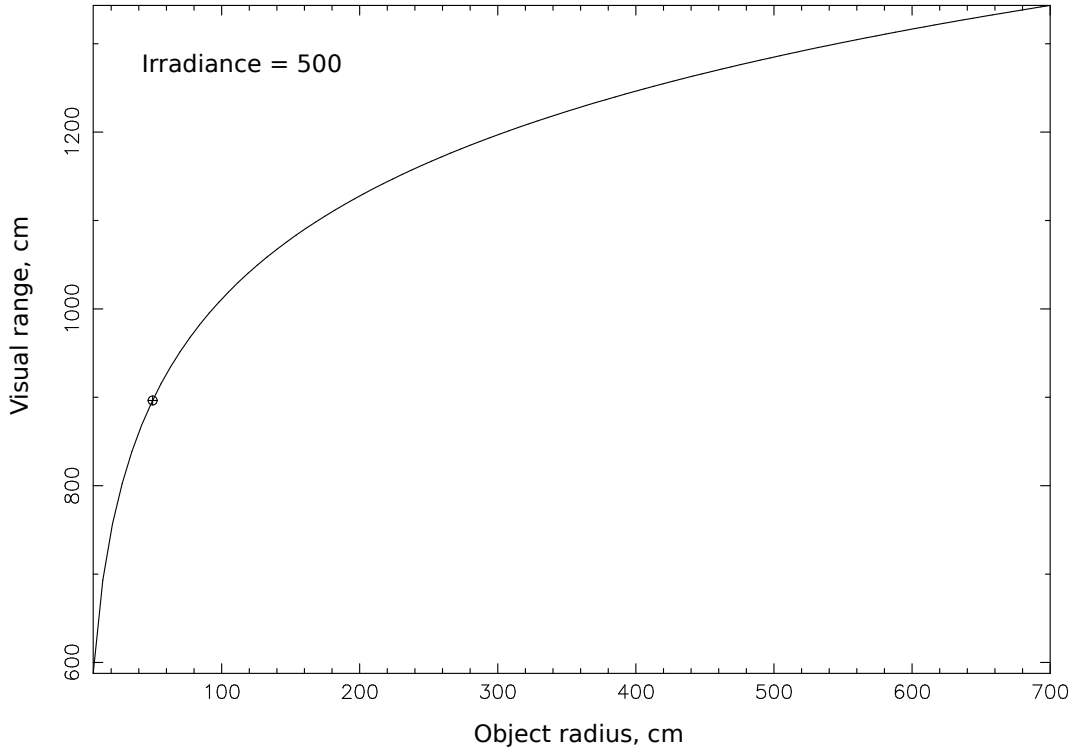
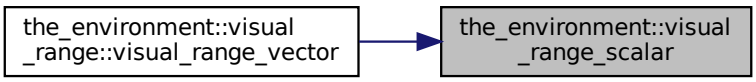


Figure 8.11 Visual range, 500

8.5.3.64.3.2 Calculation details First, check if background irradiance is below commondata::zero and just return zero visual range.
Check if prey area dummy parameter is provided, if not, use the default value `commondata::preyarea_default`.
Check if prey contrast dummy parameter is provided, if not, use the default value `commondata::preycontrast_default`.
Call the main computational backend `the_environment::srgetr()` Note that `the_environment::srgetr()`

and the whole computational backend are in commondata::hrp precision to reduce numerical rounding and avoid overflow errors.
The visual range calculation backend srgetr() seems computationally suboptimal and with large object size leads to numerical overflow, so the visual range is -Infinity. This is corrected in two steps. (1) in deriv(), a problematic part of the computation now checks proactively for potential overflow (comparing to log(huge()) for the current FPU precision level), so no NaNs or Infinity are produced and no FPU invalid arithmetic errors occur. This is reported to the logger.
If the visual range value returned by srgetr is negative, we reset it to the maximum overflow ceiling value VISUAL_RANGE_MAX_OVERFLOW. It has previously be set to easyr() approximation, but that was grossly wrong (hugely overestimating). It is safer to just limit visual range for detecting bigger objects to such a fixed value.
Finally, do explicit conversion of the final return value from the high commondata::hrp to the standard precision commondata::srp.
Definition at line 4730 of file m_env.f90.
Here is the caller graph for this function:



8.5.3.65 visual_range_vector()

```
real(srp) function, dimension(size(prey_area)), private the_environment::visual_range_vector (
            real(srp), intent(in) irradiance,
            real(srp), dimension(:), intent(in) prey_area,
            real(srp), dimension(size(prey_area)), intent(in), optional prey_contrast_vect,
            real(srp), intent(in), optional prey_contrast )  [private]
```

Wrapper for calculating *visual range of a fish predator* using the Dag Aksnes's procedures srgetr(), easyr() and deriv(). See srgetr() for computational details.

Note

This is a **vector** version, prey_area is mandatory and also defines the vector size for all other vector parameters including the returned function value vector. This is useful for selecting among a swarm of prey with different sizes when vector is processed. The measurement unit here is meter. Might need conversion if other units are used.

Parameters

in	*irradiance*	background irradiance at specific depth
in	*prey_area*	prey area, m^2; Mandatory parameter.
in	*prey_contrast_vect*	optional prey inherent contrast or default parameter if not present. This parameter sets **individual vector** prey contrast, so can be used for providing stochastic contrast data for each object.
in	*prey_contrast*	optional prey inherent contrast or default parameter if not present. This parameter sets **common scalar** prey contrast for the whole vector.

Returns

Returns visual range of the fish predator.

8.5.3.65.1 Specific implementations See specific implementations:

- the_environment::visual_range_scalar() for scalar argument
- the_environment::visual_range_vector() for vector argument
- the_environment::visual_range_fast() elemental (parallel-safe) version lacking sanity checks and extended debugging.

8.5.3.65.2 Implementation notes The computational backend for the visual range computation `srgetr()`, `easyr()` and `deriv()` now uses the `commondata::q_prec_128` 128 bit numerical precision model. This precision is sufficient to calculate visual range up to the the object radius of approximately 700 cm (area 153.9 cm^2) without FPU overflow errors. At this level, the maximum visual range is 1343.34 cm.

8.5.3.65.2.1 Visual range plots The visual range plots below are generated by `HEDTOOLS/tools/visrange_plot.f90`.

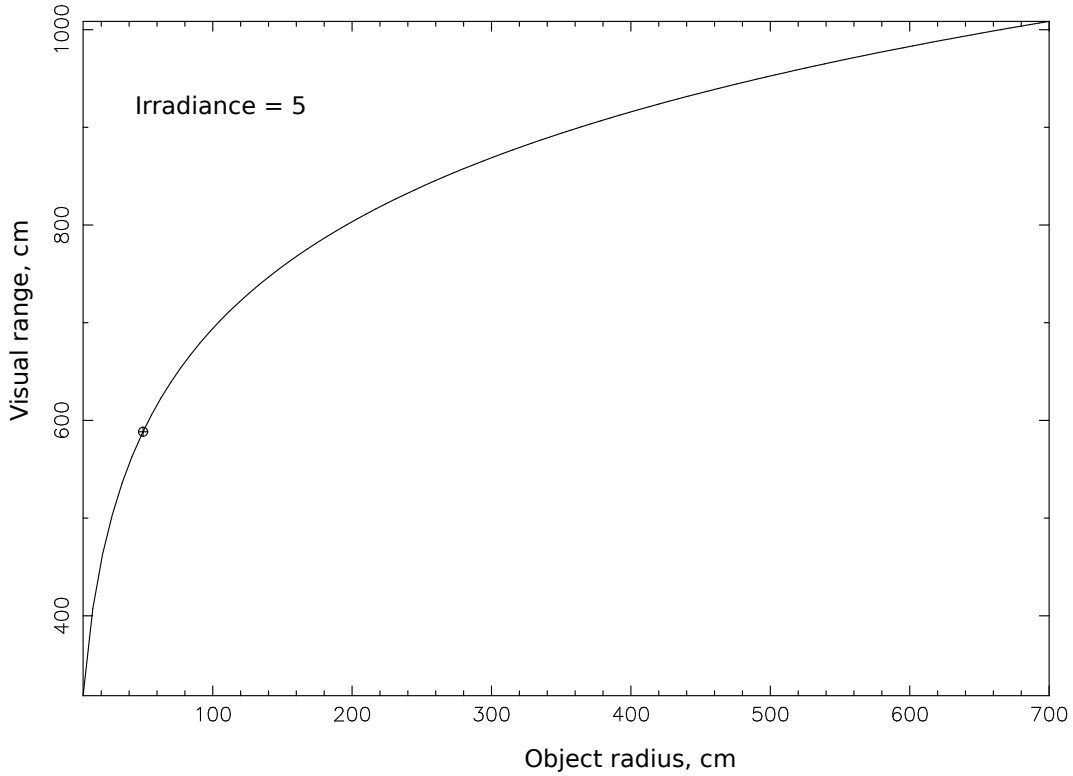


Figure 8.12 Visual range, 0.5

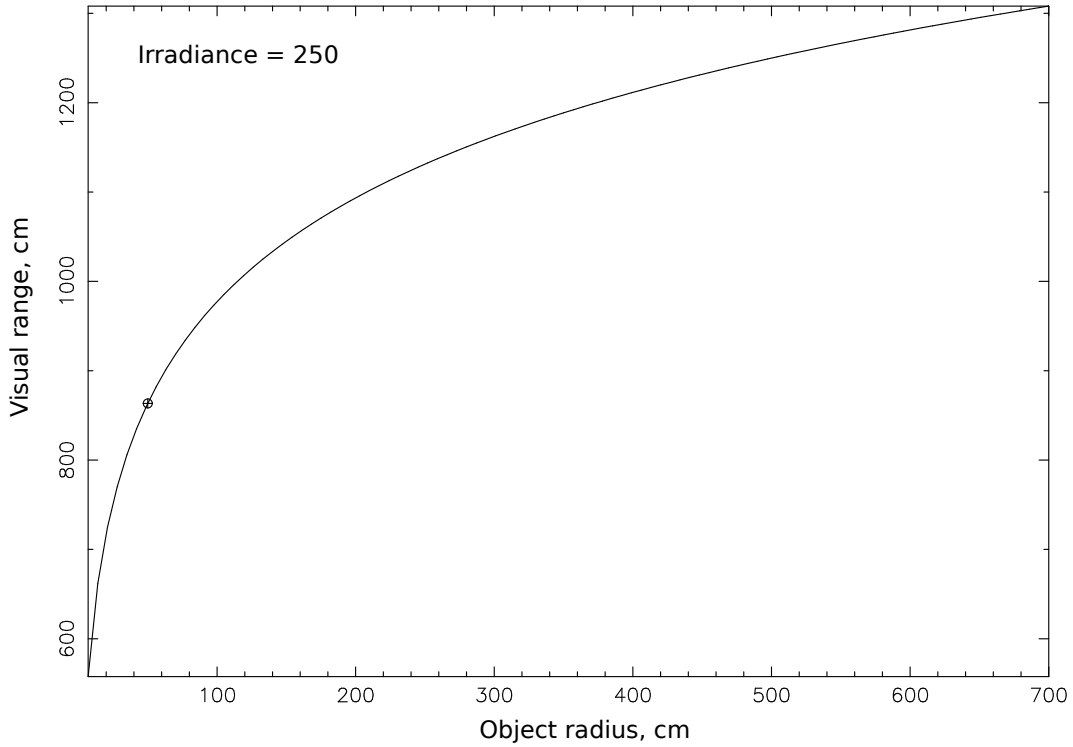


Figure 8.13 Visual range, 250

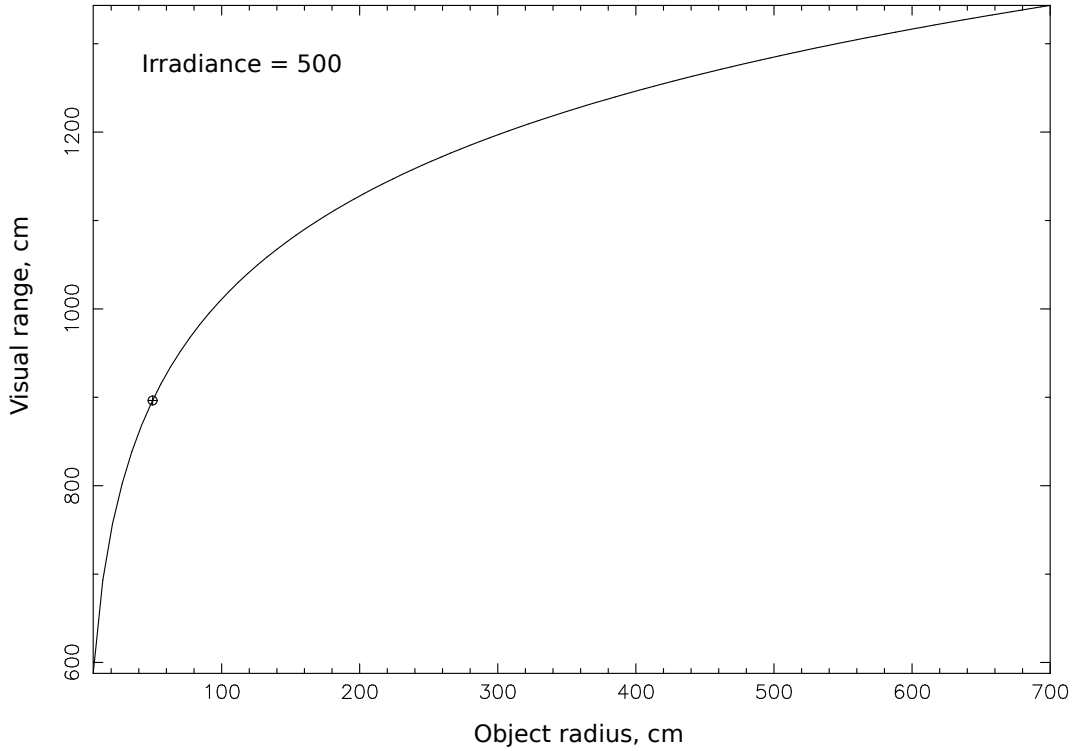


Figure 8.14 Visual range, 500

Check if prey contrast dummy parameter is provided, if not, use the default value `commondata::preycontrast_default`. Note that the function can use either a (possibly stochastic) vector or a scalar value (same for all) parameters for prey contrast as optional arguments. Scalar value takes precedence if both are provided.

8.5.3.65.2.2 Calculation of the visual range The main body of calculation is actually a loop calling the scalar-based function `the_environment::visual_range_scalar()`.
Definition at line 4898 of file m_env.f90.

8.5.3.66 visual_range_fast()

```
elemental real(srp) function the_environment::visual_range_fast (
            real(srp), intent(in) irradiance,
            real(srp), intent(in), optional prey_area,
            real(srp), intent(in), optional prey_contrast )
```

Wrapper for calculating *visual range of a fish predator* using the Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`. This is a new **elemental** and parallel-ready visual range function wrapper making use the elemental-procedures based computational backend. See notes on `visual_range_scalar()` and `srgetr()` for computational details.

Parameters

in	*irradiance*	background irradiance at specific depth
in	*prey_area*	prey area, m^2
in	*prey_contrast*	optional prey inherent contrast or default parameter if not present.

Returns

Returns visual range of the fish predator

Warning

It is simplified, e.g. **no error reporting** is done. Nonetheless, debugging the old code has shown that it works okay up to the `MAX_LOG` non-whale size limit. Use the non-elemental version whenever debugging or logging is required! The parameter `prey_contrast` to the **vector**-based function call must be an **scalar**. Otherwise a segmentation fault runtime error results. Vector-based call is analogous to calling `visual_range_vector` with `prey_contrast_vect` parameter.

8.5.3.66.1 Specific implementations See specific implementations:

- the_environment::visual_range_scalar() for scalar argument
- the_environment::visual_range_vector() for vector argument
- the_environment::visual_range_fast() elemental (parallel-safe) version lacking sanity checks and extended debugging.

8.5.3.66.2 Implementation details This version of the visual_range procedure does not call the error correction code and does not report errors into the logger. However, this allows declaring it as **elemental**.

Note

Note that `srgetr()` and the whole computational backend are now in `commondata::hrp` precision to avoid numerical overflow errors.

Finally, do explicit conversion from `commondata::hrp` to `commondata::srp`.
Definition at line 5009 of file m_env.f90.

8.5.3.67 srgetr()

```
elemental subroutine, private the_environment::srgetr (
            real(hrp), intent(out) r,
            real(hrp), intent(in) c,
```

```
real(hrp), intent(in) C0,
real(hrp), intent(in) Ap,
real(hrp), intent(in) Vc,
real(hrp), intent(in) Ke,
real(hrp), intent(in) Eb,
integer, intent(out), optional IER )  [private]
```

Obtain visual range by solving the non-linear equation by means of Newton-Raphson iteration and derivation in subroutine the_environment::deriv(). Initial value is calculated in the_environment::easyr(). The calculation is based on the model described in Aksnes & Utne (1997) Sarsia 83:137-147.

Note

Programmed and tested 29 January 2001 Dag L Aksnes.

This subroutine is left almost intact with only the most crucial changes. (a) added commondata::hrp precision specifier (128 bit precision model) to real type specifiers and _HRP for literal constants; (b) restored diagnostic IER output from archival Hed11.f90. (c) added explicit intent and declared the procedures as pure that is required for being parallel-friendly.

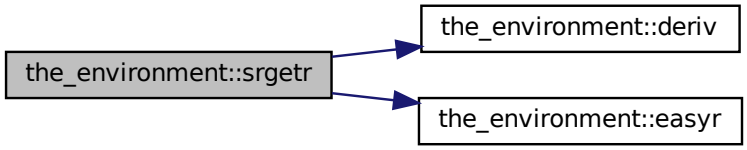
Parameters

in	*RST*	start value of r calculated by the_environment::easyr();
in	*c*	beam attenuation coefficient (m-1);
in	*C0*	prey inherent contrast;
in	*Ap*	prey area (m^2);
in	*Vc*	parameter characterising visual capacity (d.l.)
in	*this*	parameter is denoted E' in Aksnes & Utne;
in	*Ke*	saturation parameter (uE m-2 s-1);
in	*Eb*	background irradiance at depth DEPTH;
out	*r*	the predator's visual range (when F1=0);
out	*IER*	**1** = No convergence after IEND steps, error return; **2** = Return in case of zero divisor; **3** = r out of allowed range (negative); **0** = valid r returned.

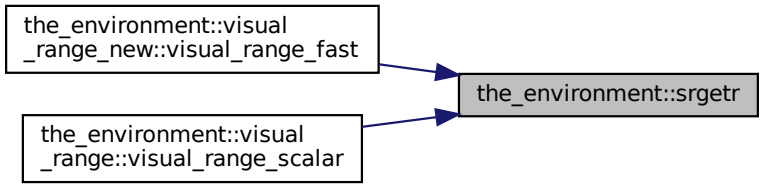
Notable variables:

- **F1** function value of equation in deriv;
- **FDER** the derivative of the function.

Definition at line 5148 of file m_env.f90.
Here is the call graph for this function:



Here is the caller graph for this function:



8.5.3.68 easyr()

```
elemental subroutine, private the_environment::easyr (
            real(hrp), intent(out) r,
            real(hrp), intent(in) C0,
            real(hrp), intent(in) Ap,
            real(hrp), intent(in) Vc,
            real(hrp), intent(in) Ke,
            real(hrp), intent(in) Eb )  [private]
```

Obtain a first estimate of visual range by using a simplified expression of visual range. See `srgetr()` for more details.

Note

This subroutine is left almost intact, only (a) added commondata::hrp for real type (`HRP` is *high real precision* (128 bit) and is defined in commondata).

Definition at line 5237 of file m_env.f90.
Here is the caller graph for this function:

8.5.3.69 deriv()

```
elemental subroutine, private the_environment::deriv (
            real(hrp), intent(inout) r,
            real(hrp), intent(out) F1,
            real(hrp), intent(out) FDER,
            real(hrp), intent(in) c,
            real(hrp), intent(in) C0,
            real(hrp), intent(in) Ap,
            real(hrp), intent(in) Vc,
```

```
            real(hrp), intent(in) Ke,
            real(hrp), intent(in) Eb )  [private]
```

Derivation of equation for visual range of a predator. See the_environment::srgetr() for more details.

Note

This is a high precision version. But higher precision alone is not sufficient to prevent numerical exponentiation overflow.

This subroutine is left almost intact, only (a) added commondata::hrp for literal constants and real type (HRP is the *high real precision* (128 bit) and is defined in commondata), (b) added numerical overflow safeguard code based on MAX_LOG exponentiability limit; added logging of overflow using LOG_MSG.

Parameters

out	*F1*	function value of equation in the_environment::deriv();
out	*FDER*	the derivative of the function;
in,out	*r*	the predator's visual range (when F1=0).

Input parameters:
See explanation in calling routine the_environment::srgetr().
The function and the derivative is calculated on the basis of the log-transformed expression.

8.5.3.69.1 Implementation notes MAX_LOG is a parameter determining the safe limit of exp function overflow in the current float point precision model, well below this. We cannot calculate precise exponent of a value exceeding this parameter. The maximum possible exponentiation-able value is set to the maximum **128-bit** real value kind Q↩_PREC_128, this bottom line value would set a safe limit for HRP calculations. A benefit of this approach is that it doesn't require IEEE exception handling that depends on not fully portable IEEE modules.

Note

The real 128 bit limit (Q_PREC_128) is sufficient to calculate visual range up to the fish length of approximately 700 cm (area 153.9 cm2). At this level, the maximum visual range is 1343.34 cm. This would be sufficient for non-whales.

Definition at line 5270 of file m_env.f90.
Here is the caller graph for this function:

8.5.3.70 light_surface_deterministic()

```
elemental real(srp) function, private the_environment::light_surface_deterministic (
            integer, intent(in), optional tstep )  [private]
```

Calculate deterministic surface light at specific time step of the model. Light (surlig) is calculated from a sine function. Light intensity just beneath the surface is modelled by assuming a 50 % loss by scattering at the surface:

$$L_t = L_{max} 0.5 sin(\pi dt/\Omega)$$

.

Returns

surface light intensity.

Parameters

tstep	time step of the model, limited by maximum commondata::lifespan.

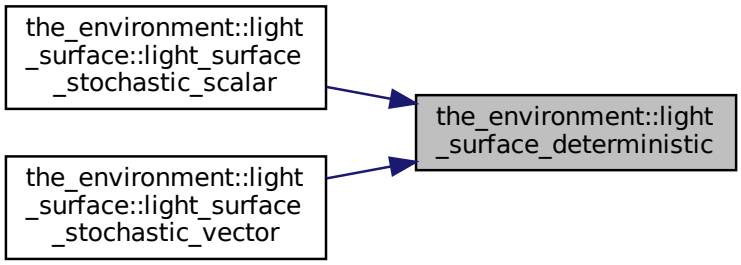
Note

This is a deterministic version. Code for wxMaxima for quickcalc:

```
surlig(a, span) := 500*0.5*(1.01+sin(3.14*2.*50*a/(1.*span)));
surlig(a, span) := 500*0.5*(1.01+sin(3.14*2.*50*a/span));
wxplot2d(surlig(a, 14000), [a,0, 1400]);
```

Note that this is an elemental function that accepts both scalar and array parameter.

Definition at line 5359 of file m_env.f90.
Here is the caller graph for this function:



8.5.3.71 light_surface_stochastic_scalar()

```
real(srp) function, private the_environment::light_surface_stochastic_scalar (
            integer, intent(in), optional tstep,
            logical, intent(in) is_stochastic )  [private]
```

Calculate stochastic surface light at specific time step of the model. Light (`surlig`) is calculated from a sine function. Light intensity just beneath the surface is modelled by assuming a 50 % loss by scattering at the surface:

$$L_t = L_{max} 0.5 sin(\pi dt/\Omega)$$

. This deterministic value sets the *mean* for the stochastic final value, which is Gaussian with CV equal to `DAYLIGHT_CV`.

Returns

surface light intensity

Parameters

tstep	time step of the model, limited by maximum commondata::lifespan.
is_stochastic	logical indicator for stochastic light intensity if TRUE, then Gaussian stochastic version is used, if FALSE, deterministic is used. Code for wxMaxima for quickcalc: `surlig(a, span) := 500*0.5*(1.01+sin(3.14*2.*50*a/(1.*span))); wxplot2d(surlig(a, 14000), [a,0, 1400]);`

Definition at line 5399 of file m_env.f90.

8.5.3.72 light_surface_stochastic_vector()

```
real(srp) function, dimension(size(tstep)), private the_environment::light_surface_stochastic↩
_vector (
            integer, dimension(:), intent(in) tstep,
            logical, intent(in) is_stochastic )  [private]
```

Calculate stochastic surface light at specific time step of the model.

Parameters

tstep	time step of the model, limited by maximum commondata::lifespan.

Returns

surface light intensity.

Parameters

is_stochastic	logical indicator for stochastic light intensity if TRUE, then Gaussian stochastic version is used, if FALSE, deterministic is used.

Note

This function accepts vector arguments.

Warning

Note that the `tstep` array parameter is *mandatory* here (otherwise the generic interface is ambiguous).

8.5.3.72.1 Implementation details If `is_stochastic` flag is TRUE, then the light intensity is a random Gaussian vector with *each element* mean equal to the deterministic value and coefficient of variation `DAYLIGHT_CV`. If `is_stochastic` flag is FALSE, deterministic *vector* value is used.
Definition at line 5447 of file m_env.f90.

8.5.3.73 light_depth_integer()

```
real(srp) function, private the_environment::light_depth_integer (
            integer, intent(in) depth,
            real(srp), intent(in), optional surface_light,
            logical, intent(in), optional is_stochastic )  [private]
```

Calculate underwater light at specific depth given specific surface light.
Underwater light is attenuated following Beer's law,

$$E_b(z,t) = L_t e^{-Kz},$$

where $E_b(z,t)$ is background irradiance at depth z at time t and K is the attenuation coefficient for downwelling irradiance. The value of K in the old code was set very high to allow the vertical dynamics to take place within 30 depth cells.

Returns

Eb background irradiance at specific depth.

Parameters

in	*depth*	The integer depth horizon where we get background
in	*surface_light*	Irradiance at the surface, normally calculated at specific time point of the model with the the_environment::light_surface() function. If this parameter is absent, surface light at the current time step is obtained. The time step in such case is obtained from commondata::global_time_step_model_current.
in	*is_stochastic*	stochastic indicator for the surface light in the_environment::light_surface() function. If this parameter is absent, the default commondata::daylight_stochastic parameter value is used.

Note

Note that this function accepts **integer** depth, a separate function should be used for physical real type depth.

Note that it is an elemental function that accepts both scalar and array parameters.

Note

Note that the commondata::lightdecay parameter is in cm.

Definition at line 5507 of file m_env.f90.

8.5.3.74 light_depth_real()

```
real(srp) function, private the_environment::light_depth_real (
            real(srp), intent(in) depth,
            real(srp), intent(in), optional surface_light,
            logical, intent(in), optional is_stochastic )  [private]
```

Calculate underwater light at specific depth given specific surface light.
Underwater light is attenuated following Beer's law,

$$E_b(z,t) = L_t e^{-Kz},$$

where $E_b(z,t)$ is background irradiance at depth z at time t and K is the attenuation coefficient for downwelling irradiance.

Returns

Eb background irradiance at specific depth.

Parameters

in	*depth*	The integer depth horizon where we get background.
in	*surface_light*	Irradiance at the surface, normally calculated at specific time point of the model with the the_environment::light_surface() function. If this parameter is absent, surface light at the current time step is obtained. The time step in such case is obtained from commondata::global_time_step_model_current.
in	*is_stochastic*	stochastic indicator for the surface light in the_environment::light_surface() function. If this parameter is absent, the default commondata::daylight_stochastic parameter value is used.

Note

Note that this function accepts **real** depth.

Note

Note that the commondata::lightdecay parameter is in cm.

wxMaxima quick code for plotting (assuming surface light 500.0):

```
wxplot2d( 500.0*exp(-0.002 * d), [d, 0., 3000.] );
```

Definition at line 5582 of file m_env.f90.

8.5.3.75 dist_scalar()

```
elemental real(srp) function the_environment::dist_scalar (
             real(srp), intent(in) x1,
             real(srp), intent(in) x2,
             real(srp), intent(in) y1,
             real(srp), intent(in) y2,
             real(srp), intent(in), optional z1,
             real(srp), intent(in), optional z2 )
```

Calculate distance between 3D or 2D points. This is a function engine for use within type bound procedures. **Example** (dist_scalar):

```
dist(1.0,10.0,   2.0,20.0,   3.0,30.0 )
```

Note

This version accepts individual scalar coordinates.

Note that it is an elemental function, accepting also arrays (should have equal size and shape for elemental operations).

Note

3D distance is calculated only if z1 and z2 are provided, otherwise an orphaned z coordinate is ignored.

Definition at line 5633 of file m_env.f90.

8.5.3.76 dist_vector_nd()

```
pure real(srp) function the_environment::dist_vector_nd (
             real(srp), dimension(:), intent(in) cvector1,
             real(srp), dimension(:), intent(in) cvector2 )
```

Calculate distance between N-dimensional points. This is a function engine for use within other type bound procedures.

Returns

the distance between two N-dimensional vectors.

Parameters

cvector	N-dimensional vectors for the two points we calculate the distance between. For a 3D case the vercors look like: x = cvector(1), y = cvector(2), z = cvector(3). **Example** dist_vector: `dist( [1.0, 2.0, 3.0], [10.0, 20.0, 30.0] )`

Note

This version accepts vectors of coordinates for each of the two objects.

Warning

The shapes and sizes of the two arrays must be equal

Definition at line 5666 of file m_env.f90.

8.5.3.77 dist2_vector()

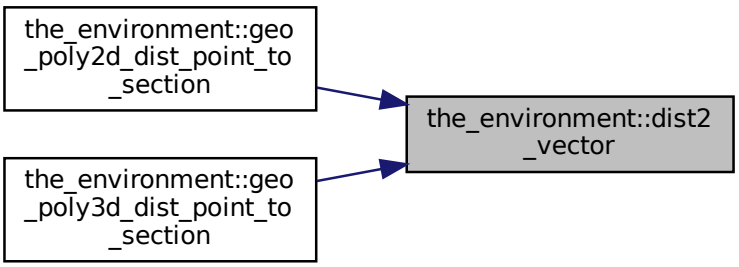
```
pure real(srp) function the_environment::dist2_vector (
             real(srp), dimension(:), intent(in) cvector1,
             real(srp), dimension(:), intent(in) cvector2 )
```

Calculate the squared distance between two *N*-dimensional points.

Note

This function is useful in some cases when squared distances are used to save on calculation of square root.

Definition at line 5683 of file m_env.f90.
Here is the caller graph for this function:



8.5.3.78 vect_magnitude()

```
pure real(srp) function the_environment::vect_magnitude (
             real(srp), dimension(:), intent(in) vector )
```

Calculate the magnitude of an arbitrary *N*-dimensional vector. This is a raw vector backend.

Parameters

in	*vector*	vector a vector in *N* dimensions.

Returns

The magnitude of the vector.

Vector length is trivially calculated as the euclidean norm:

$$\|x\| = \sum x_i^2.$$

Definition at line 5699 of file m_env.f90.

8.5.3.79 dist2step()

```
elemental real(srp) function the_environment::dist2step (
            real(srp), intent(in) average_distance,
            integer, intent(in), optional dimensionality )
```

Calculate the unit step along a single coordinate axis given the average distance between any two points in a N-dimensional Gaussian random walk.

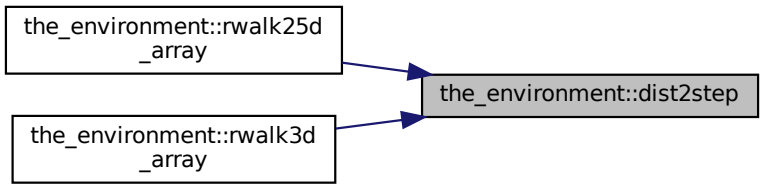
Returns

Unit step length along a single x,y,or z axis

Parameters

average_distance	The average distance traversed by the moving point during a single step of the Gaussian random walk.
dimensionality	The dimensionality of the random walk **Example:** dist2step Generate a Gaussian random walk in 3D with discrete step size, i.e. the **true distance between the points** (rather than coordinate shifts) equal to 10.0: `call moving_zzz%rwalk( dist2step(10.0), 0.5 )`

Definition at line 5724 of file m_env.f90.
Here is the caller graph for this function:



8.5.3.80 food_item_create()

```
elemental subroutine the_environment::food_item_create (
            class(food_item), intent(inout) this )
```

Create a single food item at an undefined position with default size.
This also cleanups the history stack, i.e. fills it with `MISSING` values.
Definition at line 5751 of file m_env.f90.

8.5.3.81 food_item_make()

```
elemental subroutine the_environment::food_item_make (
            class(food_item), intent(inout) this,
            type(spatial), intent(in) location,
            real(srp), intent(in), optional size,
            integer, intent(in) iid )
```

Make a single food item, i.e. place it into a specific position in the model environment space and set the size.

Parameters

Location	of the food item as a SPATIAL type container
size	This is the optional size of the food item. If absent then default food size is used as defined in COMMONDATA
iid	id for the food item. Note: There are no random iids for food items, iid should agree with the item index within the food resource object.

8.5.3.81.1 Implementation details We here just set the location of the food object using standard interface function position.
Also, clean up the history stack, i.e. fills it with MISSING values.
Then we set the food item size. Check if optional size is provided and left untouched if not.

Note

Note that if the value provided is very small (e.g. zero or below), a minimum default value is used as a "floor".

This food item is NOT eaten from creation, so set the status FALSE.
Set the individual id iid.
Definition at line 5786 of file m_env.f90.

8.5.3.82 food_item_capture_success_stochast()

```
logical function the_environment::food_item_capture_success_stochast (
            class(food_item), intent(in) this,
            real(srp), intent(in), optional prob )
```

Stochastic outcome of **this** food item capture by an agent. Returns TRUE if the food item is captured.
In this version, food item capture depends only on the **fixed probability** set by default as FOOD_ITEM_↩ CAPTURE_PROBABILITY. Could also implement more complex patterns, e.g. dependent on the food item size (e.g. capture probability increases with food item size).

Parameters

in	*prob*	fixed probability of food item capture. return@ TRUE if capture success.

Warning

This function does not change the state of the food item object, only returns success.

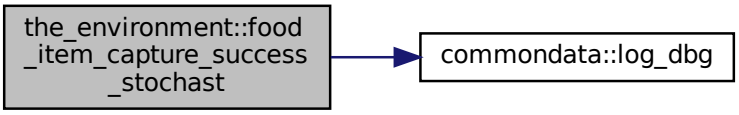
This function cannot be made elemental / puredue to random number call.

8.5.3.82.1 Implementation details Check if prob is present, if not, use default parameter FOOD_ITEM↩ _CAPTURE_PROBABILITY value, assuming this food item is in proximity of the predator agent (see function capture_probability.
First check if this food item is available, if it is not, this may mean it was an error (e.g. only available food items should get into the agent perception object).

Note

Note that the probability of capture is fixed by the input value and does not currently depend on the properties of the food item itself. TODO: make option to depend on **this** food item properties, e.g. its size.

Definition at line 5840 of file m_env.f90.
Here is the call graph for this function:



8.5.3.83 food_item_capture_probability_calc()

```
real(srp) function the_environment::food_item_capture_probability_calc (
            class(food_item), intent(in) this,
            real(srp), intent(in), optional distance,
            integer, intent(in), optional time_step_model )
```

Calculate the probability of capture of **this** food item by a predator agent depending on the distance between the agent and this food item.
Capture probability is determined on the basis of a non-parametric relationship (interpolation) between the distance between the predator agent and this food item. It is equal to the the baseline value set by `commondata::food_item_capture_probability` parameter at the distance 0.0 and approaches a nearly zero value set by the parameter `commondata::food_item_capture_probability_min = 0.1`, at the distance equal to the visual range of the agent.

Parameters

in	*distance*	optional distance to the food item.
in	*time_step_model*	optional time step of the model, if absent, obtained from the global variable `commondata::global_time_step_model_current`.

Returns

The probability of capture of this food item.

8.5.3.83.1 Implementation details

8.5.3.83.1.1 Visual range First, the visual range for a predator R_v to detect this food item is calculated using the_environment::food_item_visibility_visual_range().
The probability of capture of this food item by a predator is obtained by a nonparametric function that is based on nonlinear interpolation.

8.5.3.83.1.2 Interpolation grid abscissa The distance for calculating the probability of capture of this food item by a predator is expressed in terms of the predator's visual range. The interpolation grid abscissa is set to the distance zero, half of the visual range ($1/2R_v$) and full visual range (R_v) of the predator agent.

8.5.3.83.1.3 Interpolation grid ordinate It is assumed that the probability of capture of this food item is equal to the `commondata::food_item_capture_probability` parameter at the distance zero, and reduces to

0.8 of this value at $1/2R_v$, and further reduces to commondata::food_item_capture_probability_min = 0.1 at the full visual range distance (R_v).

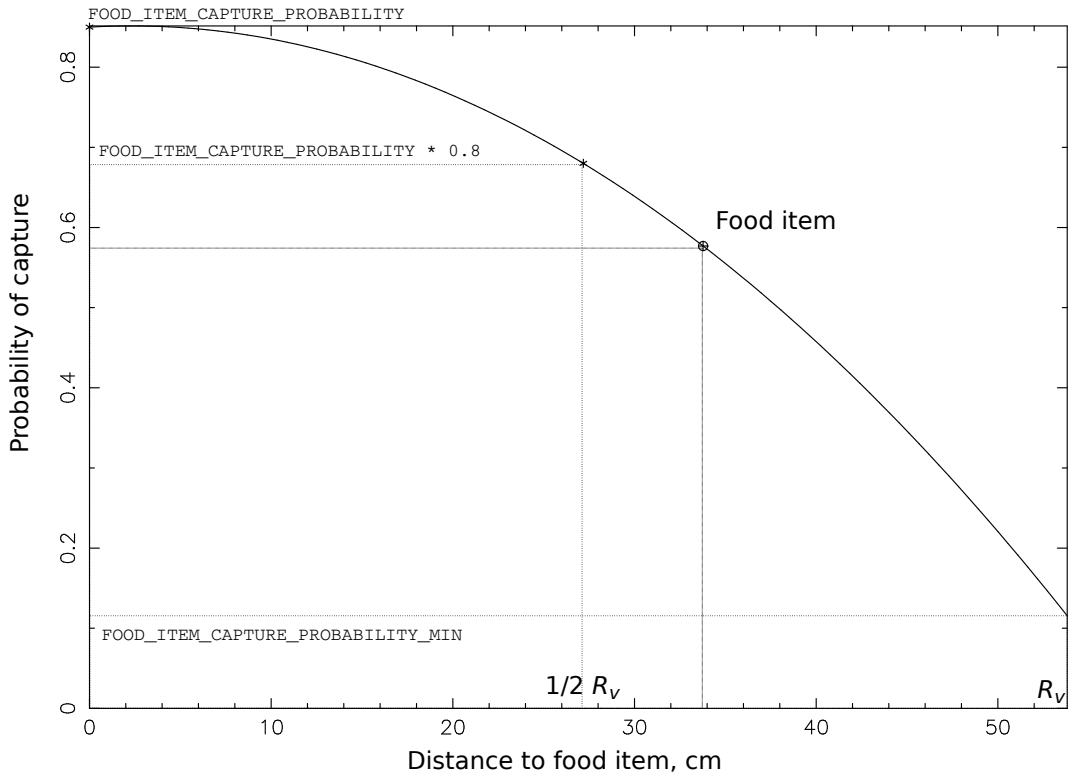


Figure 8.15 Capture probability and visual range

Note

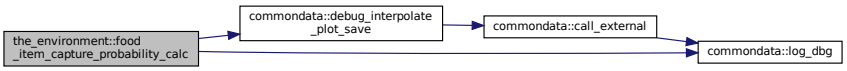
Interpolation plot command: htintrpl.exe [0.0 0.5 1.0] [0.85, 0.68, 0.1] (0.68=0.↩ 85∗0.8).

8.5.3.83.1.4 Optional distance parameter Check optional distance dummy parameter and set local value or default (half a visual range) if the parameter is not provided.
Also check if the distance provided is longer than the visual range so this food item cannot be detected by the agent. There should be normally no such cases, if this occurs, it may point to a bug.
If this is the case, set the capture probability to zero and exit.
Finally, he distance provided should not be equal to the missing value commondata::missing. Return zero capture probability in such a case.

8.5.3.83.1.5 Final calculations Finally, the probability of capture of this food item by a predator is obtained by **nonlinear** (DDPINTERPOL) interpolation of the distance value expressed in terms of the visual range over the grid set by interpol_abscissa and interpol_ordinate. There is an additional condition *0* $< p <$ *1* that is enforced by commondata::within().

8.5.3.83.1.6 Extended debugging outputs Log the capture probability visual range calculated, along with the distance and the visual range.
Interpolation plots can be saved in the debug mode using this plotting command: commondata::debug_interpolate_plot_save().
Definition at line 5897 of file m_env.f90.

Here is the call graph for this function:



8.5.3.84 food_item_visibility_visual_range()

```
real(srp) function the_environment::food_item_visibility_visual_range (
             class(food_item), intent(in) this,
             real(srp), intent(in), optional object_area,
             real(srp), intent(in), optional contrast,
             integer, intent(in), optional time_step_model )
```

Calculate the visibility range of this food item. Wrapper to the the_environment::visual_range() function. This function calculates the distance from which this food item can be seen by a predator (i.e. the default predator's visual range).

Warning

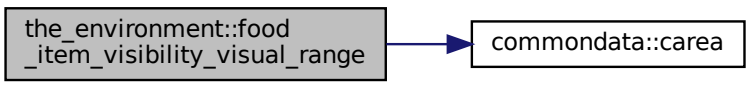
The `visual_range` procedures use meter for units, this auto-converts to cm.

Cannot implement a generic function accepting also vectors of this objects as only elemental object-bound array functions are allowed by the standard. This function cannot be elemental, so passed-object dummy argument must always be scalar.

Parameters

in	*object_area*	object_area is optional area of the food item (m). If not provided, obtained from the this object `size` attribute (the_environment::food_item::size).
in	*contrast*	contrast is optional inherent visual contrast of the food item. the default contrast of all objects is defined by the commondata::preycontrast_default parameter.
in	*time_step_model*	optional time step of the model, if absent gets the current time step as defined by the value of `commondata::global_time_step_model_current`.

8.5.3.84.1 Implementation details Calculate ambient illumination / irradiance at the depth of this food item at the given time step.
Return visual range of a predator to see this food item.
Definition at line 6036 of file m_env.f90.
Here is the call graph for this function:



8.5.3.85 minimum_depth_visibility()

```
real(srp) function the_environment::minimum_depth_visibility (
            real(srp), intent(in) target_range,
            real(srp), intent(in), optional depth_range_min,
            real(srp), intent(in), optional depth_range_max,
            real(srp), intent(in), optional object_area,
            real(srp), intent(in), optional object_contrast,
            integer, intent(in), optional time_step_model )
```

Find the depth at which the visibility of a spatial object becomes smaller than a specific distance value `target_range`.

Note

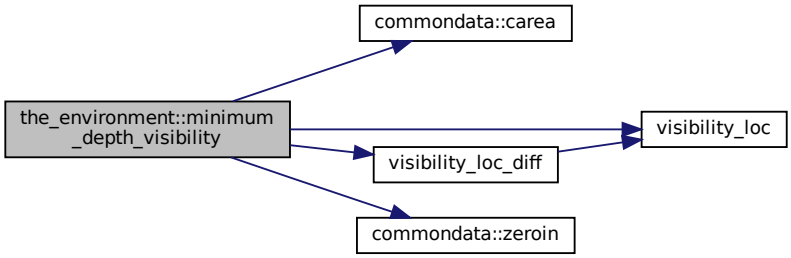
This is a diagnostic function.

Parameters

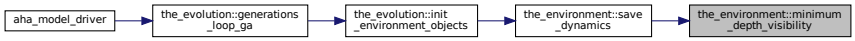
in	*target_range*	target_range This is the target visual range (visibility) value: this function calculates the depth at which visibility becomes smaller than this target range.
in	*depth_range_min*	!>
in	*depth_range_min*	sets the optimisation range for depth, this is the **minimum** depth where the search is done. This is an optional parameter, if absent, is set to the global minimum depth in the global habitats array commondata::global_habitats_available.
in	*depth_range_max*	!>
in	*depth_range_max*	sets the optimisation range for depth, this is the **maximum** depth where the search is done. This is an optional parameter, if absent, is set to the global maximum depth in the global habitats array commondata::global_habitats_available.
in	*object_area*	object_area is the optional area of the spatial object (m) for which the depth is calculated. If absent, is calculated for the default average food item commondata::food_item_size_default.
in	*object_contrast*	object_contrast optional contrast of the spatial object for which the calculation is done; if absent is set from the default commondata::preycontrast_default.
in	*time_step_model*	time_step_model is the time step of the model. If absent, is obtained from the global variable commondata::global_time_step_model_current.

8.5.3.85.1 Implementation details The depth when the visibility of the spatial object becomes smaller than the target distance `target_range` is calculated using the Brent's commondata::zeroin() optimisation algorithm, see Brent, R., (1973). Algorithms for Minimization Without Derivatives, Prentice-Hall, Inc.
The function for calculating the visibility of this spatial object for optimisation by `zeroin` is calculated by visibility_loc(). This function is local to this function and is further wrapped into also local visibility_loc_diff() (it is set as the parameter f to commondata::zeroin()).
If the the depth cannot be calculated, further checks are done and an appropriate limiting value is set for this function return.
Definition at line 6100 of file m_env.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



8.5.3.86 food_item_disappear()

```
elemental subroutine the_environment::food_item_disappear (
            class(food_item), intent(inout) this )
```

Make the food item "disappear" and take the "eaten" state, i.e. impossible for consumption by the agents.
Definition at line 6253 of file m_env.f90.

8.5.3.87 food_item_is_eaten_unavailable()

```
elemental logical function the_environment::food_item_is_eaten_unavailable (
            class(food_item), intent(in) this )
```

Logical check-indicator function for the food item being eaten and not available.

Returns

Logical indicator TRUE if the food item is eaten and not available any more.

Definition at line 6265 of file m_env.f90.

8.5.3.88 food_item_is_available()

```
elemental logical function the_environment::food_item_is_available (
            class(food_item), intent(in) this )
```

Logical check-indicator function for the food item being available.

Returns

Logical indicator TRUE if the food item is present in the environment and therefore available.

Note

It is the opposite of the above function the_environment::food_item::is_unavailable().

Definition at line 6286 of file m_env.f90.

8.5.3.89 food_item_get_size()

```
elemental real(srp) function the_environment::food_item_get_size (
            class(food_item), intent(in) this )
```

Get the size component of the food item object.

Returns

item_size The size of the food item.

Definition at line 6304 of file m_env.f90.

8.5.3.90 size2mass_food()

```
elemental real(srp) function the_environment::size2mass_food (
            real(srp), intent(in) radius )
```

Calculate the mass of a food item, the non-OO backend.
The food item mass depends on the density of the food and its volume

$$\rho \cdot \frac{4}{3}\pi r^3,$$

where ρ is the food density and

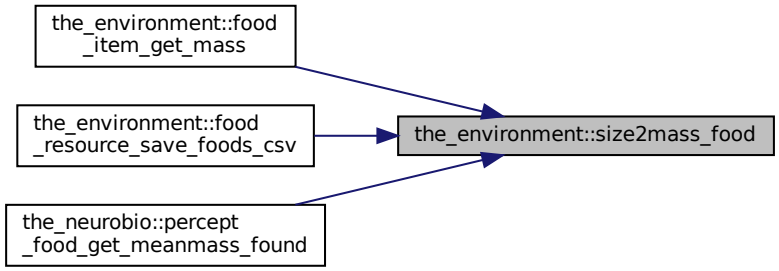
$$V = \frac{4}{3}\pi r^3$$

is the food item volume.

Note

Food item density is set by commondata::food_item_density parameter.

Definition at line 6322 of file m_env.f90.
Here is the caller graph for this function:



8.5.3.91 mass2size_food()

```
elemental real(srp) function the_environment::mass2size_food (
            real(srp), intent(in) mass )
```

Calculate the size (radius) of a food item, a reverse function of the_environment::size2mass_food():

$$= \sqrt[3]{\frac{M}{\rho \cdot 4/3\pi}},$$

where M is the food item mass, ρ is its density.

Note

Food item density is set by commondata::food_item_density parameter.

Definition at line 6337 of file m_env.f90.

8.5.3.92 food_item_get_mass()

```
elemental real(srp) function the_environment::food_item_get_mass (
            class(food_item), intent(in) this )
```

Calculate and get the mass of the food item.

Returns

item_mass The mass of the food item.

Note

This is an OO frontend/wrapper.

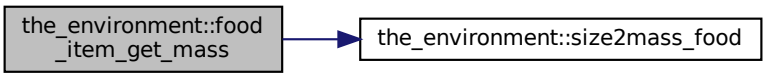
8.5.3.92.1 Implementation details The food item mass depends on the density of the food and its volume

$$\rho \cdot \frac{4}{3}\pi r^3,$$

where ρ is the food density and

$$V = \frac{4}{3}\pi r^3$$

is the food item volume.
Definition at line 6349 of file m_env.f90.
Here is the call graph for this function:



8.5.3.93 food_item_set_iid()

```
elemental subroutine the_environment::food_item_set_iid (
            class(food_item), intent(inout) this,
            integer, intent(in) iid )
```

Set unique id for the food item object.

Parameters

iid	individual id number for the food item.

Definition at line 6367 of file m_env.f90.

8.5.3.94 food_item_clone_assign()

```
elemental subroutine the_environment::food_item_clone_assign (
```

```
            class(food_item), intent(in) this,
            class(food_item), intent(inout) the_other )
```

Clone the properties of this food item to another food item.

Note

Note that the this food item serves as the source and the other, as the destination for cloning. So when used like this `food_item_sourceclone( cloned_to_this_destination_item )`

Parameters

`in,out`	*the_other*	param[inout] the_other Target food item to which the properties of the **this** item are cloned. It is declared as a polymorphic `class` so can get an extension type.

8.5.3.94.1 Implementation details Use make method to transfer all the properties from `this` to `the_other`.
Definition at line 6383 of file m_env.f90.

8.5.3.95 food_item_get_iid()

```
elemental integer function the_environment::food_item_get_iid (
            class(food_item), intent(in) this )
```

Get the unique id of the food item object.

Returns

iid the individual id number of this food item.

Definition at line 6401 of file m_env.f90.

8.5.3.96 food_resource_make()

```
pure subroutine the_environment::food_resource_make (
            class(food_resource), intent(inout) this,
            character(len=*), intent(in) label,
            integer, intent(in) abundance,
            type(spatial), dimension(:), intent(in) locations,
            real(srp), dimension(:), intent(in) sizes )
```

Make food resource object. This class standard constructor.

Parameters

label	Label for this food resource object.
abundance	the number of food items in the resource.
locations	An array of `SPATIAL` locations of each food item.
sizes	An array of sizes of each food item.

8.5.3.96.1 Implementation details First, we allocate the array of the food item objects with the size of `abundance` parameter.
Set label from input data.
Set abundance from input data.
Create all food items in the resource,
Set locations and food item sizes from input data vectors.
Definition at line 6417 of file m_env.f90.

8.5.3.97 food_resource_get_abundance()

```
elemental integer function the_environment::food_resource_get_abundance (
            class(food_resource), intent(in) this )
```

Get the number of food items in the food resource.

Returns

The number of food items in this food resource.

Definition at line 6465 of file m_env.f90.

8.5.3.98 food_resource_get_label()

```
elemental character(len=label_length) function the_environment::food_resource_get_label (
            class(food_resource), intent(in) this )
```

Get the label of the this food resource.

Returns

The label of this food resource.

Definition at line 6476 of file m_env.f90.

8.5.3.99 food_resource_destroy_deallocate()

```
pure subroutine the_environment::food_resource_destroy_deallocate (
            class(food_resource), intent(inout) this )
```

Delete and deallocate food resource object. This class standard destructor.
Definition at line 6487 of file m_env.f90.

8.5.3.100 food_resource_locate_3d()

```
pure type(spatial) function, dimension(size(this%food)) the_environment::food_resource_↩
locate_3d (
            class(food_resource), intent(in) this )
```

Get the location object array (array of SPATIAL objects) of a food resource object.

Returns

locate_array an array of SPATIAL location objects for all food items within the resource.

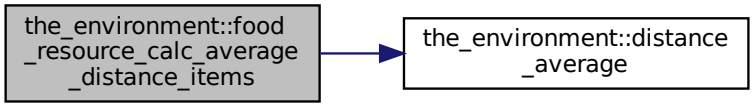
Definition at line 6506 of file m_env.f90.

8.5.3.101 food_resource_calc_average_distance_items()

```
real(srp) function the_environment::food_resource_calc_average_distance_items (
            class(food_resource), intent(in) this,
            integer, intent(in), optional n_sample )
```

Calculate the average distance between food items within a resource. e.g. to compare it with the agent's random walk step size.
Definition at line 6527 of file m_env.f90.

Here is the call graph for this function:



8.5.3.102 food_resource_replenish_food_items_all()

```
subroutine the_environment::food_resource_replenish_food_items_all (
            class(food_resource), intent(inout) this,
            integer, intent(in), optional replace )
```

Replenish and restore food resource. The food resource is replenished by substituting randomly selected `replace` food items or all items if `replace` is omitted or exceeds the actual number of food items. Unlike the the_environment::food_resource::make() method, the sizes and the positions of the food items within the resource are reused from the previous positions (previously explicitly set set by the the_environment::food_resource::make() method).

Warning

This method is not used to build the food resource for the first time ("init"). Use the_environment::food_resource::make() to do this instead. This method is only for *modifying* the existing food resource object.

Parameters

in	*replace*	replace Optional number of food items that replace those those that have been eaten in the population, this replace number cannot make the food resource population size bigger than the previous value.

8.5.3.102.1 Implementation notes Implementation differs depending on full (`replase` absent) or partial replenishment (`replace` present).

- If the `replace` parameter is set, a maximum of this number of eaten food items will be returned to the food resource, i.e. set the "not eaten" status.

First count food items that are eaten.

- If the number of food items replaced (`replace`) exceeds this value `n_eaten` we reset all eaten food items as "not eaten."
- Otherwise, the number of food items defined by `replace` will be replaced randomly (set not eaten) in the eaten. To do this:
- allocate a temporary array of the size equal to this number of eaten items, initialised to commondata::unknown.
- put indexes of all eaten food items into the `idx_eaten_items` array.
- reorder `idx_eaten_items` in a random order
- finally, reset the first `replace` food items as not eaten
- If `replace` parameter is not present, just reset the the_environment::food_resource::food::eaten indicator of each food item to FALSE. Thus, the food resource is just reset to a fully available state.

Note

Note that the food items within the food resource retain their sizes and positions unchanged. If fully stochastic resource is required or its size needs to be altered, use the the_environment::food_resource::make() method.

Definition at line 6561 of file m_env.f90.

8.5.3.103 food_resource_migrate_move_items()

```
subroutine the_environment::food_resource_migrate_move_items (
             class(food_resource), intent(inout) this,
             real(srp), intent(in), optional max_depth,
             integer, intent(in), optional time_step_model )
```

This subroutine implements the migration of all the food items in the resource according to the plankton migration pattern from the G1 model (HED18). Briefly, the movement of each of the food items has two components:

- deterministic: regular vertical migration movement;
- stochastic: small scale random Gaussian displacement.

8.5.3.103.0.1 Global habitats array If the habitats are assembled into the global array the_environment::global_habitats_available and the migration is done on the original habitat-bound food resource objects, these original habitat objects and the global array require constant synchronisation with the the_environment::assemble() and the_environment::disassemble() procedures. Here is an example:

```
call disassemble( habitat_test1, habitat_test2 )
call habitat_test1%food%migrate_vertical()
call habitat_test2%food%migrate_vertical()
call assemble ( habitat_test1, habitat_test2 )
```

To avoid this, a separate procedure that works directly on an *array* of habitat objects is implemented↩ : the_environment::migrate_food_vertical(). Its use is more efficient:

```
call migrate_food_vertical( Global_Habitats_Available )
```

Parameters

in	*max_depth*	max_depth optional maximum depth of deterministic vertical migration; if this parameter is absent, the maximum depth in the habitat container for this food resource is used
in	*time_step_model*	time_step_model Optional time step of the model. If absent, the time step is obtained from the global array commondata::global_habitats_available.

8.5.3.103.1 Implementation notes

8.5.3.103.1.1 Checks and preparations First, the procedure checks if the maximum depth for the vertical migration of food items is provided. If not, the maximum depth is determined as the maximum depth in the habitat of a single randomly chosen food item in this resource (see the_environment::spatial::find_environment()). Because a single food resource is nested within a specific habitat, all items are located in this environment, so the maximum depth is the same for all items of the resource.
Thus, `habitat_res` keeps the number of the habitat in the global array the_environment::global_habitats_available
Second, a check is done if the time step is provided. If no, the global time step from commondata::global_time_step_model_current is used.

8.5.3.103.1.2 Deterministic move The depth centre of the food item vertical distribution is determined using the code from the HED18 model with minimum adaptations. This code is isolated into the the_environment::center_depth_sinusoidal() function.
Once the target depth for the food items is known, all the food resource is moved up or down to the target depth.

8.5.3.103.1.3 Stochastic walk The second phase of the food item migration involves a stochastic random walk of each of the food items. This stochastic movement is described by the following parameters of the

the_environment::spatial_moving::rwalk():

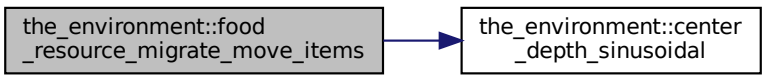
- commondata::food_item_migrate_xy_mean,
- commondata::food_item_migrate_xy_cv ,
- commondata::food_item_migrate_depth_mean,
- commondata::food_item_migrate_depth_cv.
- the limiting habitat for random walk is defined by the `habitat_res` element of the global array the_environment::global_habitats_available the same for all items as food resource is nested within the habitat.

Note

Note that the stochastic random walk coincides with the default the_environment::food_resource::rwalk() method. The code is retained independently here because the parameters might not coincide with the default walk. Otherwise use call the default

```
call this%rwalk()
```

Definition at line 6660 of file m_env.f90.
Here is the call graph for this function:



8.5.3.104 food_resource_rwalk_items_default()

```
subroutine the_environment::food_resource_rwalk_items_default (
            class(food_resource), intent(inout) this )
```

Perform a random walk step for all food items within the food resource. The walk is performed with the default parameters:

- commondata::food_item_migrate_xy_mean,
- commondata::food_item_migrate_xy_cv ,
- commondata::food_item_migrate_depth_mean,
- commondata::food_item_migrate_depth_cv.

8.5.3.104.1 Implementation notes First, the habitat number within the global habitats array the_environment::global_habitats_avai is determined for a single randomly chosen food item within this resource. Because the food resource is bound to the habitat, this identifies the habitat.
Then, all food items within the resource are subjected to Gaussian random walk the_environment::spatial_moving::rwalk() with the following parameters:

- commondata::food_item_migrate_xy_mean,
- commondata::food_item_migrate_xy_cv ,
- commondata::food_item_migrate_depth_mean,
- commondata::food_item_migrate_depth_cv.

- the limiting habitat for random walk is defined by the `habitat_res` element of the global array the_environment::global_habitats_available the same for all items as food resource is nested within the habitat.

Definition at line 6769 of file m_env.f90.

8.5.3.105 migrate_food_vertical()

```
subroutine the_environment::migrate_food_vertical (
             class(habitat), dimension(:), intent(inout) habitats,
             integer, intent(in), optional time_step_model )
```

Migrate food items in a whole array of food resources. The array is normally the the_environment::global_habitats_available.

```
call migrate_food_vertical( Global_Habitats_Available )
```

Note

This is a not type-bound procedure.

8.5.3.105.0.1 Global habitats array All the habitat objects are normally assembled into the global array the_environment::global_habitats_available. If so, this procedure makes it unnecessary to synchronise the habitat objects with the global array constantly by calling the_environment::habitat::disassemble() and the_environment↩ ::habitat::assemble() procedures whenever the habitat objects are changed (foods migrated). It makes the migration change directly on the *global array*.
The the_environment::food_resource::migrate_vertical() method that operates directly on *food resource object* in such case would require constant synchronisation/update between the global array and each habitat-bound food resource object, e.g.:

```
call disassemble( habitat_test1, habitat_test2 )
call habitat_test1%food%migrate_vertical()
call habitat_test2%food%migrate_vertical()
call assemble ( habitat_test1, habitat_test2 )
```

Parameters

`in,out`	*habitats*	[inout] habitats is an array of habitats
`in`	*time_step_model*	time_step_model Optional time step of the model. If absent, the time step is obtained from the global array commondata::global_time_step_model_current.

8.5.3.105.1 Implementation notes A check is done if the time step is provided. If no, the global time step from commondata::global_time_step_model_current is used.
Then food resources within each of the habitats within the `habitats` array is subjected to the the_environment::food_resource::migra method.
Definition at line 6834 of file m_env.f90.

8.5.3.106 rwalk_food_step()

```
subroutine the_environment::rwalk_food_step (
             class(habitat), dimension(:), intent(inout) habitats )
```

Perform a random walk of food items in a whole array of food resources. The array is normally the the_environment::global_habitats_available. This procedure is a wrapper for the_environment::food_resource::rwalk() to do a walk on a whole array of habitats and linked food resources.

```
call rwalk_food_step( Global_Habitats_Available )
```

Note

This is a not type-bound procedure.

Parameters

in,out	*habitats*	[inout] habitats is an array of habitats

8.5.3.106.1 Implementation notes All the food resources within each of the habitats within the `habitats` array is subjected to the the_environment::food_resource::rwalk() method.
Definition at line 6874 of file m_env.f90.

8.5.3.107 center_depth_sinusoidal()

```
real(srp) function the_environment::center_depth_sinusoidal (
              integer, intent(in) tstep,
              real(srp), intent(in) depth )
```

This function calculates the target depth for the sinusoidal vertical migration pattern of the food items at each time step of the model. See the_environment::food_resource_migrate_move_items() for the calling procedure.

Note

The depth centre of the food item vertical distribution is determined using the code from the HED18 model with minimum adaptations. This function isolates this HED18 code. The pattern coincides with the COPDVM = 1 in the HED18 model. Different pattern(s) is easy to add in separate procedures.

This procedure is isolated into a separate one, so diagnostic data/plots can be saved.

Parameters

in	*tstep*	tstep time step of the model.
in	*depth*	depth sets the maximum target depth for vertical migration.

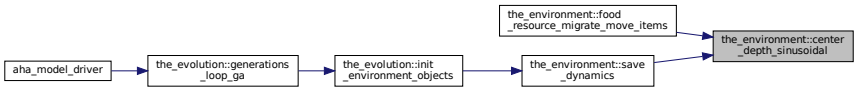
Returns

The target depth for the regular migration of the food items.

Verbatim code from HED18:

```
do a = 1, flifespan
  !find center of copepod distribution
  copsindepth = sin(3.14*a*2.*dielcycles/(flifespan+1.))
  copcenterdepth = int(depth/2 + 0.33*depth*copsindepth)
```

Definition at line 6902 of file m_env.f90.
Here is the caller graph for this function:

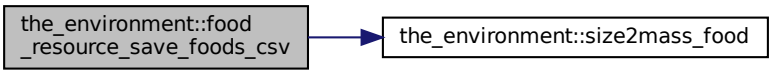


8.5.3.108 food_resource_save_foods_csv()

```
subroutine the_environment::food_resource_save_foods_csv (
              class(food_resource), intent(in) this,
              character(len=*), intent(in), optional csv_file_name,
              logical, intent(out), optional is_success )
```

Save characteristics of food items in the resource into a CSV file.

8.5.3.108.1 Implementation notes First, check if the optional CSV file name is provided, if not, generate it automatically.
Then, data for the food resource is saved using the `CSV_MATRIX_WRITE()` procedure from `HEDTOOLS`.
The CSV output data file can be optionally compressed with the commondata::cmd_zip_output command if commondata::is_zip_outputs is set to TRUE.
Definition at line 6929 of file m_env.f90.
Here is the call graph for this function:



8.5.3.109 food_resource_sort_by_size()

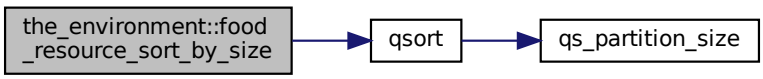
```
elemental subroutine the_environment::food_resource_sort_by_size (
            class(food_resource), intent(inout) this,
            logical, intent(in), optional reindex )
```

Sort the food resource objects within the array by their sizes. The two subroutines below are a variant of the recursive quick-sort algorithm adapted for sorting real components of the the `FOOD_RESOURCE` object.

Parameters

in	*reindex*	is a logical flag enabling re-indexing the food resource after it is sorted. The default is **NOT** to reindex.

Definition at line 6990 of file m_env.f90.
Here is the call graph for this function:



8.5.3.110 food_resource_reset_iid_all()

```
pure subroutine the_environment::food_resource_reset_iid_all (
            class(food_resource), intent(inout) this,
            integer, intent(in), optional start_iid )
```

Reset individual iid for the food resource. Individual iids must normally coincide with the array order index. If it is changed after sorting, iids no longer reflect the correct index. So this subroutine resets iids to be coinciding with each food item index.

Parameters

start_iid	is an optional parameters that sets the starting value for reindexing. For example, indexing from 1000 rather than 1.

Warning

Always reindex food resource after food items have been sorted!

Definition at line 7085 of file m_env.f90.

8.5.3.111 reindex_food_resources()

```
subroutine the_environment::reindex_food_resources (
             class(food_resource), intent(inout), optional resource_1,
             class(food_resource), intent(inout), optional resource_2,
             class(food_resource), intent(inout), optional resource_3,
             class(food_resource), intent(inout), optional resource_4,
             class(food_resource), intent(inout), optional resource_5,
             class(food_resource), intent(inout), optional resource_6,
             class(food_resource), intent(inout), optional resource_7,
             class(food_resource), intent(inout), optional resource_8,
             class(food_resource), intent(inout), optional resource_9,
             class(food_resource), intent(inout), optional resource_10,
             class(food_resource), intent(inout), optional resource_11,
             class(food_resource), intent(inout), optional resource_12,
             class(food_resource), intent(inout), optional resource_13,
             class(food_resource), intent(inout), optional resource_14,
             class(food_resource), intent(inout), optional resource_15,
             class(food_resource), intent(inout), optional resource_16,
             class(food_resource), intent(inout), optional resource_17,
             class(food_resource), intent(inout), optional resource_18,
             class(food_resource), intent(inout), optional resource_19,
             class(food_resource), intent(inout), optional resource_20 )
```

Reset and reindex iids for an input list of several food resources. As the result of this subroutine all food items across all the resources within the whole list will have unique iids.

Parameters

in,out	*resource_1,resource_2,...*	a list of food resources to reindex

Note

The calculation does not use an array of food resources because this can create problems in setting input dummy parameters in the array constructor. It just accepts raw resource objects and does all the operations directly on them. The number of food resources is probably never big, so the hard-coded limit of 20 components would probably never be exceeded. But the object list implementation is rather wordy, code-duplicating and prone to editing bugs. **The main aim** of this wordy, code-dubbing and mistype-prone approach was to allow easy passage of the whole original resource objects back from the collapsed object retaining all the changes that were introduced (e.g. the `eaten` status) while the resource objects were processed as the joined super-object. TODO: Perhaps it could be reimplemented in a better style using an extension object type.

Warning

Note that this is not a type-bound subroutine. It should not be declared in the `FOOD_RESOURCE` type.

Note

See notes on food_resources_collapse(), food_resources_update_back() and reindex_food_resources().

Definition at line 7135 of file m_env.f90.

8.5.3.112 food_resources_collapse()

```
subroutine the_environment::food_resources_collapse (
             class(food_resource), intent(out) food_resource_collapsed,
             class(food_resource), intent(in), optional resource_1,
             class(food_resource), intent(in), optional resource_2,
             class(food_resource), intent(in), optional resource_3,
             class(food_resource), intent(in), optional resource_4,
             class(food_resource), intent(in), optional resource_5,
             class(food_resource), intent(in), optional resource_6,
             class(food_resource), intent(in), optional resource_7,
             class(food_resource), intent(in), optional resource_8,
             class(food_resource), intent(in), optional resource_9,
             class(food_resource), intent(in), optional resource_10,
             class(food_resource), intent(in), optional resource_11,
             class(food_resource), intent(in), optional resource_12,
             class(food_resource), intent(in), optional resource_13,
             class(food_resource), intent(in), optional resource_14,
             class(food_resource), intent(in), optional resource_15,
             class(food_resource), intent(in), optional resource_16,
             class(food_resource), intent(in), optional resource_17,
             class(food_resource), intent(in), optional resource_18,
             class(food_resource), intent(in), optional resource_19,
             class(food_resource), intent(in), optional resource_20,
             logical, intent(in), optional reindex,
             character(len=*), intent(in), optional label )
```

Collapse several food resources into one. The collapsed resource can then go into the perception system. The properties of the component resources are retained in the collapsed resource.

Parameters

out	*food_resource_collapsed*	output resource object that is formed by joining the list of component resource objects.
in	*resource_1,resource_2,...*	a list of component resource objects.
in	*reindex*	logical flag to reindex the joined resource (TRUE) upon joining. The default is **no** reindexing.
in	*label*	Label for the joined food resource, if absent set to 'tmp_object'.

8.5.3.112.1 Implementation notes The calculations in this procedure does not use an array of food resources because this can create problems in setting input dummy parameters in the array constructor. It just accepts raw resource objects and does all the operations directly on them. The number of food resources is probably never big, so the hard-coded limit of 20 components would probably never be exceeded. But the object list implementation is rather wordy, code-duplicating and prone to editing bugs. **The main aim** of this wordy, code-dubbing and mistype-prone approach was to allow easy passage of the whole original resource objects back from the collapsed object retaining all the changes that were introduced (e.g. the eaten status) while the resource objects were processed as the joined super-object. TODO: Perhaps it could be reimplemented in a better style using an extension object type. The joined food resource object retains the original component ID's. If common unique iids are needed, reindex method should be called upon joining! This is performed now using the optional 'reindex' parameter. If the food items are re-sorted within the joined food resource, their iid's no longer correspond to the original id's.

Therefore, unjoining will result in totally new order and id's and the stacking of the original component resources is **broken**: correct unjoining is then impossible.

8.5.3.112.2 Usage example This example shows how to use the `join` and `unjoin` methods to collapse and split back food resources.

```
! Join two resources into 'joined_food_res_tmp'
call joined_food_res_tmp%join( habitat_safe%food,                &
                               habitat_dangerous%food,           &
                               reindex=.true. )
! We can then work with the collapsed resource, e.g. get
! the perception and use the "eat" method.
call proto_parents%individual(ind)%see_food(joined_food_res_tmp)
call proto_parents%individual(ind)%do_eat_food_item(             &
                        food_item_selected, joined_food_res_tmp)
! after this we use the "unjoin" method to get back the
! original component food resources, they are now in an updated
! state, e.g. the "eaten" flag is transferred from the collapsed
! food resource object back to the component objects.
call joined_food_res_tmp%unjoin( habitat_safe%food,              &
                                 habitat_dangerous%food,         &
                                 reindex=.true. )
! destroy the temporary collapsed resource object.
call joined_food_res_tmp%destroy()
```

Note

See notes on `food_resources_collapse()`, `food_resources_update_back()` and `reindex_food_resources()`.

8.5.3.112.3 Implementation details For each food resource that is provided in the list we take the temporary transfer arrays for parameter copying from the component food resource into the `food_resource_collapsed`. Make this food resource from the component objects that are provided.
And add extra object component arrays that do not get modified by `make`: the **eaten** status and the id number (**iid**).
If the `reindex` flag is present and is TRUE, do reindexing the new joined food resource.
Definition at line 7350 of file m_env.f90.

8.5.3.113 food_resources_collapse_global_object()

```
type(food_resource) function the_environment::food_resources_collapse_global_object (
             logical, intent(in), optional reindex,
             character(len=*), intent(in), optional label )
```

Join food resources into a single global food resource out of the global array the_environment::global_habitats_available. See the_environment::unjoin() for how to unjoin an array of food resources back into an array. The joined resource can then go into the perception system. The properties of the component resources are retained in the collapsed resource.

Note

This procedure is intended to use a short interface name `join`, see the_environment::join().

A similar procedure using a **list** of input component resources is implemented in the_environment::food_resources_collapse().

Parameters

in	*reindex*	reindex logical flag to reindex the joined resource (TRUE) upon joining. The default is **no** reindexing.
in	*label*	label Label for the joined food resource, if absent set to 'tmp_object'.

Returns

A collapsed food resource joining the input array.

8.5.3.113.1 Implementation details This procedure builds the output `food_resource_collapsed` resource object from scratch fully substituting the normal `make` the_environment::food_resource::make() method. Check out the main method the_environment::food_resource::make() in case of reimplementation.

- Determine the total number of food items in the collapsed food resource, it equals to the sum of the items across all components of the the_environment::global_habitats_available array.

Make a label for the collapsed object, if not present as a dummy parameter, set to 'tmp_object'.

- Allocate the food items array for the collapsed food resource.
- Set the abundance for the new output object equal to the sum across all the components.
- Create all the food items in the new object using the the_environment::food_item::create() method.
- Finally, copy individual food items from the component resources into the new joined resource looping over all resources and appending arrays.
- If the `reindex` flag is present and is TRUE, do reindexing the new joined food resource.

Definition at line 7914 of file m_env.f90.

8.5.3.114 food_resources_update_back()

```
subroutine the_environment::food_resources_update_back (
             class(food_resource), intent(in) food_resource_collapsed,
             class(food_resource), intent(inout), optional resource_1,
             class(food_resource), intent(inout), optional resource_2,
             class(food_resource), intent(inout), optional resource_3,
             class(food_resource), intent(inout), optional resource_4,
             class(food_resource), intent(inout), optional resource_5,
             class(food_resource), intent(inout), optional resource_6,
             class(food_resource), intent(inout), optional resource_7,
             class(food_resource), intent(inout), optional resource_8,
             class(food_resource), intent(inout), optional resource_9,
             class(food_resource), intent(inout), optional resource_10,
             class(food_resource), intent(inout), optional resource_11,
             class(food_resource), intent(inout), optional resource_12,
             class(food_resource), intent(inout), optional resource_13,
             class(food_resource), intent(inout), optional resource_14,
             class(food_resource), intent(inout), optional resource_15,
             class(food_resource), intent(inout), optional resource_16,
             class(food_resource), intent(inout), optional resource_17,
             class(food_resource), intent(inout), optional resource_18,
             class(food_resource), intent(inout), optional resource_19,
             class(food_resource), intent(inout), optional resource_20,
             logical, intent(in), optional reindex )
```

Transfer back the resulting food resources into their original objects out from a collapsed object from `food_↩ resources_collapse`.

Parameters

`in,out`	*resource_1,resource_2,...*	a list of food resources to restore from the joined state.
`in`	*reindex*	logical flag to reindex the joined resource (TRUE) upon joining. Default is **no** reindexing.

Note

The calculation does not use an array of food resources because this can create problems in setting input dummy parameters in the array constructor. It just accepts raw resource objects and does all the operations directly on them. The number of food resources is probably never big, so the hard-coded limit of 20 components would probably never be exceeded. But the object list implementation is rather wordy, code-duplicating and prone to editing bugs. **The main aim** of this wordy, code-dubbing and mistype-prone approach was to

allow easy passage of the whole original resource objects back from the collapsed object retaining all the changes that were introduced (e.g. the `eaten` status) while the resource objects were processed as the joined super-object. TODO: Perhaps it could be reimplemented in a better style using an extension object type.

Warning

The un-joined food resource objects retain the joined object ID's. If individual id indexing are required, `reindex` method should be called for each of the unjoined food resource object upon joining!

Note

See notes on `food_resources_collapse()`, `food_resources_update_back()` and `reindex_food_resources()`.

8.5.3.114.1 Implementation details For each food item within the resource we copy all the object component values from the **collapsed** object to the original component object.
If `reindex` is explicitly set to TRUE, we reindex the component resources upon unjoining.
Definition at line 8023 of file m_env.f90.

8.5.3.115 food_resources_update_back_global_object()

```
subroutine the_environment::food_resources_update_back_global_object (
            type(food_resource), intent(in) food_resource_collapsed,
            logical, intent(in), optional reindex )
```

Transfer the (having been modified) food resource objects from the single united object `food_resource_collapsed` back to the global array the_environment::global_habitats_available array. See the_environment::join() for how to join an array of food resources into a single global object.

Parameters

in	*food_resource_collapsed*	A collapsed food resource previously joining the input array.
in	*reindex*	reindex logical flag to reindex the unjoined resource (TRUE) upon unjoining. The default is **no** reindexing.

8.5.3.115.1 Implementation details This procedure restores individual food resources into the global array the_environment::global_habitats_available array from the collapsed resource `food_resource_collapsed`.
For each food item within the resource we copy all the object component values from the **collapsed** object to the original component object.
If `reindex` is explicitly set to TRUE, we reindex the component resources upon unjoining.
Definition at line 8636 of file m_env.f90.

8.5.3.116 global_habitats_assemble()

```
subroutine the_environment::global_habitats_assemble (
            type(habitat), intent(in), optional habitat_1,
            type(habitat), intent(in), optional habitat_2,
            type(habitat), intent(in), optional habitat_3,
            type(habitat), intent(in), optional habitat_4,
            type(habitat), intent(in), optional habitat_5,
            type(habitat), intent(in), optional habitat_6,
            type(habitat), intent(in), optional habitat_7,
            type(habitat), intent(in), optional habitat_8,
            type(habitat), intent(in), optional habitat_9,
            type(habitat), intent(in), optional habitat_10,
```

```
type(habitat), intent(in), optional habitat_11,
type(habitat), intent(in), optional habitat_12,
type(habitat), intent(in), optional habitat_13,
type(habitat), intent(in), optional habitat_14,
type(habitat), intent(in), optional habitat_15,
type(habitat), intent(in), optional habitat_16,
type(habitat), intent(in), optional habitat_17,
type(habitat), intent(in), optional habitat_18,
type(habitat), intent(in), optional habitat_19,
type(habitat), intent(in), optional habitat_20,
logical, intent(in), optional reindex )
```

Assemble the global habitats objects array the_environment::global_habitats_available from a list of separate habitat objects. This call.

```
assemble(hab_a, hab_b, hab_c)
```

is equivalent to

```
global_habitats_available = [ hab_a, hab_b, hab_c ]
```

Note

But note that the `reindex` parameter allows automatic reindexing of the global array the_environment::global_habitats_available

Parameters

in	*habitat_1*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_2*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_3*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_4*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_5*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_6*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_7*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_8*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_9*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_10*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_11*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_12*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_13*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_14*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_15*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_16*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_17*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_18*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_19*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_20*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*reindex*	reindex logical flag to reindex the global joined food resource array (TRUE) linked to each of the habitats upon assemble. Default is **no** reindexing.

8.5.3.116.1 Implementation notes

- Stage 1: Calculate how many habitat objects are there in the input parameter list.

Stage 2: Allocate the the_environment::global_habitats_available global array of habitat objects the above number of elements.

- Stage 3: Build the global array of habitat objects one by one from the input list of individual habitat objects.
- Stage 4: Optionally reindex each element of the the global array the_environment::global_habitats_available.

Definition at line 8701 of file m_env.f90.

8.5.3.117 global_habitats_disassemble()

```
subroutine the_environment::global_habitats_disassemble (
            type(habitat), intent(out) habitat_1,
            type(habitat), intent(out), optional habitat_2,
            type(habitat), intent(out), optional habitat_3,
            type(habitat), intent(out), optional habitat_4,
            type(habitat), intent(out), optional habitat_5,
            type(habitat), intent(out), optional habitat_6,
            type(habitat), intent(out), optional habitat_7,
            type(habitat), intent(out), optional habitat_8,
            type(habitat), intent(out), optional habitat_9,
            type(habitat), intent(out), optional habitat_10,
            type(habitat), intent(out), optional habitat_11,
            type(habitat), intent(out), optional habitat_12,
            type(habitat), intent(out), optional habitat_13,
            type(habitat), intent(out), optional habitat_14,
            type(habitat), intent(out), optional habitat_15,
            type(habitat), intent(out), optional habitat_16,
            type(habitat), intent(out), optional habitat_17,
            type(habitat), intent(out), optional habitat_18,
            type(habitat), intent(out), optional habitat_19,
            type(habitat), intent(out), optional habitat_20,
            logical, intent(in), optional reindex )
```

Disassemble the global habitats objects array the_environment::global_habitats_available into separate habitat object.

Parameters

out	*habitat_1*	[inout] habitat_1, ... a list (from 2 to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*reindex*	reindex logical flag to reindex the joined food resource (TRUE) linked to each of the habitats upon disassemble. Default is **no** reindexing.

Definition at line 8972 of file m_env.f90.

8.5.3.118 spatial_neighbours_distances()

```
subroutine the_environment::spatial_neighbours_distances (
            class(spatial), intent(in) this,
            class(spatial), dimension(:), intent(in) neighbours,
            real(srp), dimension(:), intent(out), optional dist,
            integer, dimension(:), intent(out), optional index_vector,
            integer, dimension(:), intent(out), optional ranks,
            integer, intent(in), optional rank_max,
            logical, intent(out), optional error_flag )
```

Calculate the distances between **this** spatial object and an array of its neighbours. Optionally output the distances, sorting index vector and rankings vector for each of these neighbours. Optionally do only partial indexing, up to the order rank_max for computational speed. Procedure ARRAY_INDEX() from HEDTOOLS is used as the computational backend for partial segmented indexing.

Parameters

in	*neighbours*	an array of spatial objects that we sort by distance from **this** target object.
out	*dist*	optional vector of the distance between each of the neighbours and **this** spatial object.
out	*index_vector*	a vector for sort order indexing of the neighbours. see documentation for ARRAY_INDEX() in HEDTOOLS for more details on sort order indexing.
out	*ranks*	optional vector of rank ordering scores of each of the distances.
in	*rank_max*	sets the maximum limit on the objects to rank/index we are interested in, i.e. for partial indexing (see manual for ARRAY_INDEX).
out	*error_flag*	optional error flag, normally should be FALSE.

Warning

Cannot be made pure because of I/O calls.

8.5.3.118.1 Implementation details First, calculate the distances between this object ant all of its neighbours. This is done using the parallel do concurrent construct from F2008.
Iterative vector sorting and ranking indexes can be slow to calculate when there are many neighbours. So we need to know are they really necessary (parameters present). Also check that all vectors have the same sizes.
Partial indexing is used if rank_max parameter is provided. This will avoid full indexing of all objects which may be much faster for big arrays.
Then calculate ranks if we need them.
If we need ranks, calculate both index vector and ranks
Use partial indexing if rank_max parameter is provided. This will avoid full indexing of all objects which may be much faster for big arrays.
Full indexing otherwise.
Definition at line 9216 of file m_env.f90.

8.5.3.119 predator_make_init()

```
elemental subroutine the_environment::predator_make_init (
            class(predator), intent(inout) this,
            real(srp), intent(in) body_size,
            real(srp), intent(in) attack_rate,
            type(spatial), intent(in), optional position,
            character(len=*), intent(in), optional label )
```

Initialise a predator object.

Parameters

body_size	the body size of the predator.
attack_rate	baseline attack rate of the predator.
environment	The environment within which the predator is located initially.
label	optional label for the predator.

8.5.3.119.1 Implementation details We first create an empty spatial sub-object for the predator.
Set the body size parameter, with a limitation that it must exceed zero.
Set the capture attack rate parameter, limited to be within the range [0:1].
Set the initial position if it is provided (will remain MISSING as initialised in create method above otherwise).
Finally, set label for the predator if provided, empty if absent.
Definition at line 9328 of file m_env.f90.

8.5.3.120 predator_label_set()

```
subroutine the_environment::predator_label_set (
            class(predator), intent(inout) this,
            character(len=*), optional label )
```

Set label for the predator, if not provided, set it random.

Parameters

label	optional label for the predator.

Definition at line 9370 of file m_env.f90.

8.5.3.121 predator_get_body_size()

```
elemental real(srp) function the_environment::predator_get_body_size (
            class(predator), intent(in) this )
```

Accessor function for the predator body size (length).
Definition at line 9386 of file m_env.f90.

8.5.3.122 predator_get_attack_rate()

```
elemental real(srp) function the_environment::predator_get_attack_rate (
            class(predator), intent(in) this )
```

Accessor function for the predator attack rate.
Definition at line 9396 of file m_env.f90.

8.5.3.123 predator_capture_risk_calculate_fish()

```
real(srp) function the_environment::predator_capture_risk_calculate_fish (
            class(predator), intent(in) this,
            class(spatial), intent(in) prey_spatial,
            real(srp), intent(in) prey_length,
            real(srp), intent(in), optional prey_distance,
            logical, intent(in), optional is_freezing,
            integer, intent(in), optional time_step_model,
            character(len=*), intent(in), optional debug_plot_file )
```

Calculates the risk of capture of the fish with the spatial location defined by `prey_spatial` and the body length equal to `prey_length`. This is a backend function bound to the predator rather than prey object.

Note

This procedure calculates the probability of capture from the the predator object side:
`predator%risk_fish( agent%location, agent%get_length() )`

It is not possible for the predator-object-bound function to determine the properties of the prey agent (that is the dummy parameter) as these parameters are defined in the_neurobio module later in the class hierarchy. For example, body mass of the agent is set in the_body::condition whereas the perception object bound function the_neurobio::perception::has_pred is defined in the the_neurobio::perception class. This is why the properties of the prey, the length and the distance, are set via mandatory dummy parameters. The capture probability should normally be calculated for the agent the other way round using the frontend function the_neurobio::perception::risk_pred():
`agent%risk_pred( predator )`

Parameters

in	*prey_spatial*	prey_spatial the spatial position of a fish/agent prey.
in	*prey_length*	prey_length the length of the prey fish agent.

Parameters

in	*prey_distance*	prey_distance optional distance between the `this` predator and the prey fish agent.
in	*is_freezing*	is_freezing Optional logical indicator that the prey fish agent is immobile.
in	*time_step_model*	time_step_model optional time step of the model.
in	*debug_plot_file*	debug_plot_file optional file name for the debug nonparametric function (interpolation) plot.

Returns

The probability of successful capture of the prey by the `this` predator

8.5.3.123.1 Check optional parameters Check optional time step parameter. If unset, use global variable `commondata::global_time_step_model_current`.
Check the distance to the prey dummy parameter. If present, the dummy parameter for the distance is used. However, if the dummy parameter is not provided, the distance between the `this` predator and the prey spatial object is calculated using the spatial::distance() function.
Check optional debug plot file name, if absent, a random name is generated based on the predator's iid. However, the prey agent's iid cannot yet be determined as it is defined in the following module layers. Threfore, it should be provided, if necessary, via the optional file name parameter.

8.5.3.123.2 Implementation details

8.5.3.123.2.1 Calculate the visibility of the prey fish object First, calculate the **illumination** (background irradiance) at the depth of the prey spatial object.
Second, calculate the **fish prey area** (m) that will later go into the visual range calculation backend engine.
Third, calculate the visual range for the predator for detecting the fish prey object defined by the `prey_spatial` spatial object. It is assumed here that the predator is much larger than the agent prey.

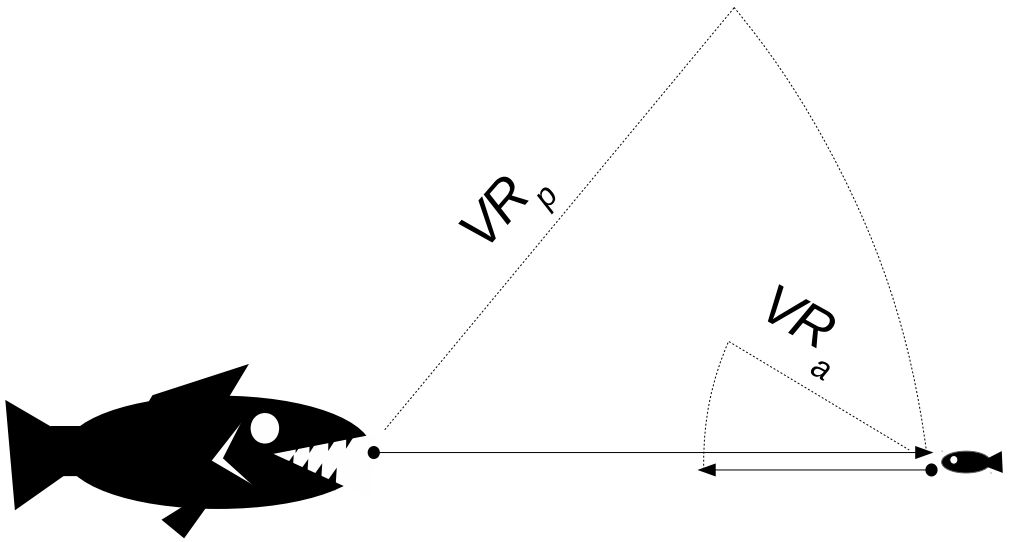


Figure 8.16 Visual ranges for the agent and the predator

At the figure above, VR_p is the visual range for the agent to detect the predator and VR_a is the visual range for the predator to see the agent prey. This is why he calculations of the capture probability are based on the latter, VR_a, the visibility of the prey for the predator.

8.5.3.123.2.2 Check prey agent visibility Check if the prey agent is visible to the predator. Predator can only attack a prey agent that it can see. Otherwise zero predation risk is returned

8.5.3.123.2.3 Calculate the predation risk Calculation is conducted differently depending on whether the agent is **moving** or **immobile** (freezing). Because the dimensionality of the interpolation grid arrays can be different in these two cases (requiring different declarations) they are isolated into two Fortran named block constructs.

Freezing (immobile) agent Calculations of the predation risk for a freezing agent are conducted in the named block construct `FREEZING`.

- Calculate the interpolation grid that is then used to calculate the predation risk.
  - The interpolation **abscissa** is defined by the visual range: from zero, half, 0.75 of the visual range and a full visual range. It is assumed that tha ability of the predator to locate and attack an immobile (freezing) agent is much smaller than a moving agent.
- The interpolation grid **ordinate** determines the nonparametric relationship between the distance between the predator and prey and the predation risk, i.e. the probability of successful capture of the prey fish agent. The probability of capture at the zero distance is fixed and equal to the predator's inherent **attack rate** (the the_environment::predator::attack_rate data component) that is always less than the theoretically possible probability 1.0, whereas at the end of the visual range, the probability of capture is zero. Therefore, the interpolation function is defined by the two middle points: 0.5 and 0.75 of the visual range:
  - commondata::predator_attack_capture_prob_frz_50
  - commondata::predator_attack_capture_prob_frz_75
- Finally, the probability of capture of the target prey fish agent is a nonparametric function of the distance between the predator and the prey calculated using the above grid arrays. There is an additional condition that this probability must be $0 < p < 1$ that is enforced by commondata::within().

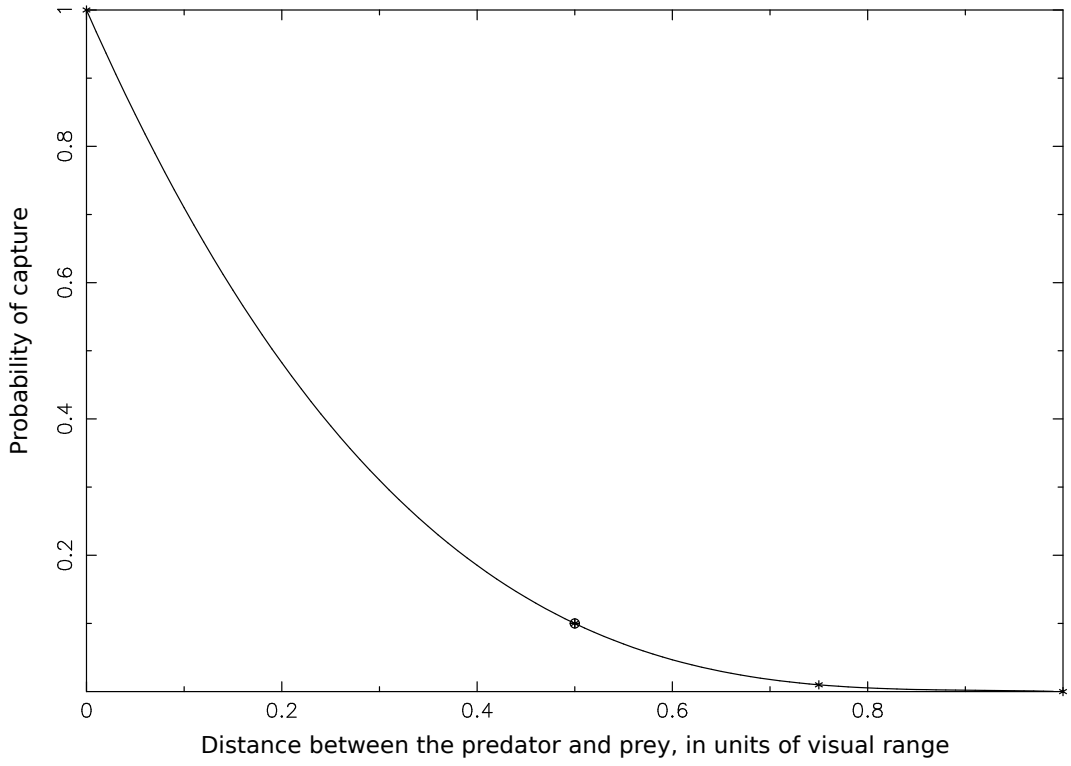


Figure 8.17 Predator capture probability of a freezing agent

Produce plot with:

```
htintrpl.exe  [0 0.5 0.75 1] [1 0.1 0.01 0] [0.5] [nonlinear]
```

- Interpolation plots can be saved in the debug mode using this plotting command: commondata::debug_interpolate_plot_save().

Normal moving agent Calculations of the predation risk for the normal moving agent are conducted in the named block construct `NORMAL_MOVING`.

- Calculate the interpolation grid that is then used to calculate the predation risk.
  - The interpolation **abscissa** is defined by the visual range: zero, distance half of the visual range, full visual range.
- The interpolation grid **ordinate** determines the nonparametric relationship between the distance between the predator and prey and the predation risk, i.e. the probability of successful capture of the prey fish agent. The probability of capture at the zero distance is fixed and equal to the predator's inherent **attack rate** (the the_environment::predator::attack_rate data component) that is always less than the theoretically possible probability 1.0, whereas at the end of the visual range, the probability of capture is much lower and is defined by the commondata::predator_attack_capture_probability_min parameter. Therefore, there is only one value that can be set independently: the probability of capture at the distance equal to 1/2 of the visual range: it is defined as the parameter constant commondata::predator_attack_capture_probability_half.
- Finally, the probability of capture of the target prey fish agent is a nonparametric function of the distance between the predator and the prey calculated using the above grid arrays. There is an additional condition that this probability must be $0 < p < 1$ that is enforced by commondata::within().

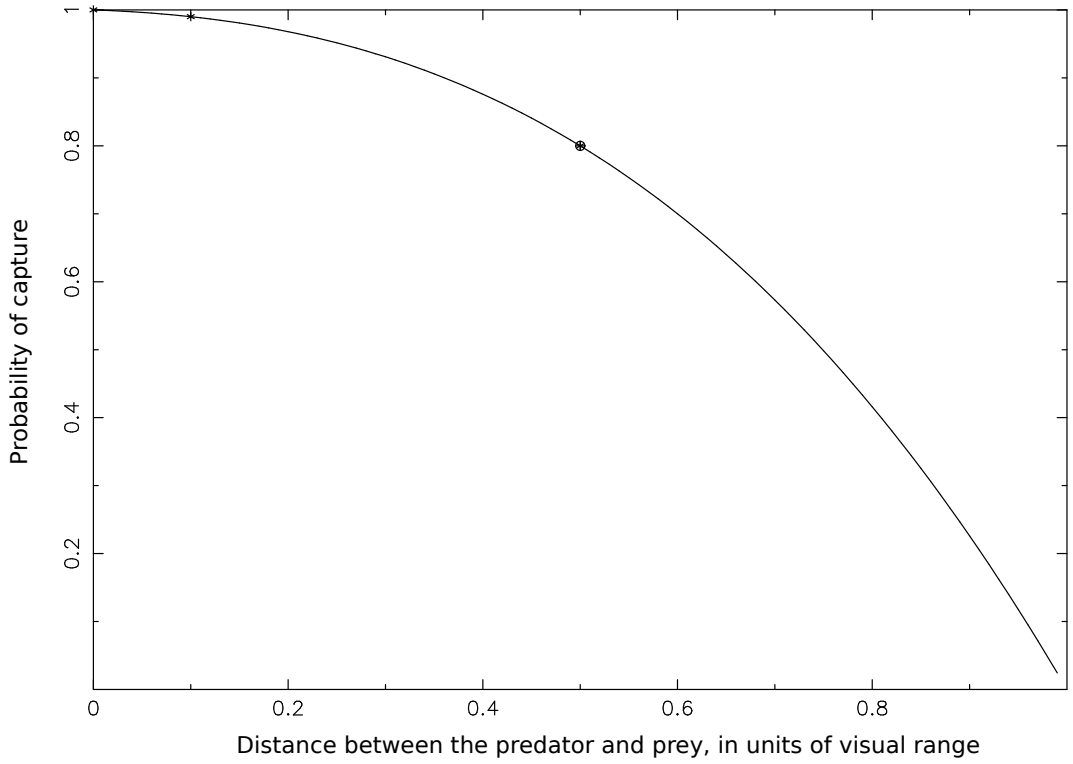


Figure 8.18 Predator capture probability

Produce plot with:

```
htintrpl.exe  [0 0.5 0.75 1] [1 0.1 0.01 0] [0.5] [nonlinear]
```

- Interpolation plots can be saved in the debug mode using this plotting command: commondata::debug_interpolate_plot_save().

8.5.3.123.2.4 Extended debugging outputs Log the capture probability visual range calculated, along with the distance and the visual range.
Definition at line 9428 of file m_env.f90.

8.5.3.124 predator_capture_risk_calculate_fish_group()

```
subroutine the_environment::predator_capture_risk_calculate_fish_group (
            class(predator), intent(in) this,
            class(spatial), dimension(:), intent(in) prey_spatial,
            real(srp), dimension(:), intent(in) prey_length,
            logical, dimension(:), intent(in), optional is_freezing,
            integer, intent(in), optional time_step_model,
            real(srp), dimension(:), intent(out), optional risk,
            real(srp), dimension(:), intent(out), optional risk_indexed,
            integer, dimension(:), intent(out), optional index_dist )
```

Calculates the risk of capture by a specific predator of an array of the fish agents with the spatial locations array defined by `prey_spatial` and the body length array `prey_length`. This subroutine takes account of both the predator dilution and confusion effects and risk adjusted by the distance towards the predator.

Note

This procedure accepts prey agents as a series of separate component arrays: `prey_spatial`, `prey_length`, `is_freezing` rather than a sigle the_individual::individual_agent type/class. This is done because the agent class hierarchy is defined specifically in the upstream modules and is not yet accessible at this level. The procedure here has the bare minimum requirements for the prey agents: the class the_environment::spatial.

Parameters

in	*prey_spatial*	prey_spatial the spatial position of a fish/agent prey.
in	*prey_length*	prey_length the length of the prey fish agent.
in	*is_freezing*	is_freezing Optional logical indicator that the prey fish agent is immobile.
in	*time_step_model*	time_step_model optional time step of the model.
out	*risk*	risk is an optional array of predation risk estimates for each agent in the group. The values of the risk for all agents that are not visible to the predator get zero risk. Note that this array has the normal order of the objects as in the input arrays, i.e. as in the normal array of the agents.
out	*risk_indexed*	risk_indexed is an optional array of predation risk estimates for each agent in the group. Risk for all agents that are not visible to the predator get zero risk. Note that this array is in the "raw" order, as the distances between the predator and the agents (not the normal order of the agents within the input group). Therefore, to translate this "raw" order it to the normal order, one requires the partial index array `index_dist`. The nature of the partial indexing dictates that only values with the index in the range from 1 to commondata::predator_risk_group_select_index_partial, i.e. the nearest neighbours of the predator make sense. All other index values are commondata::unknown.

Warning

The size of this array cannot be smaller than the partial indexing parameter commondata::predator_risk_group_select_index_par

Parameters

out	*index_dist*	index_dist optional partial index array that shows the partial sorting of the agents with respect to their distance to the predator. Note that The nature of the partial indexing dictates that only values with the index in the range from 1 to commondata::predator_risk_group_select_index_partial, i.e. the nearest neighbours of the predator make sense. All other index values are commondata::unknown.

Warning

The size of this array cannot be smaller than the partial indexing parameter commondata::predator_risk_group_select_index_par

8.5.3.124.1 Implementation details

8.5.3.124.1.1 Alternative indexing schemes Two kinds of indexing of the output adjusted risk are possible to get: normal and "raw". The array in the normal order (`risk`) is provided with the same indexing as the input arrays (e.g. the array of the prey agents. The array in the "raw" order `risk_indexed` is arranged in the order of the distances between the predator and the prey agents (the first is the nearest). The latter "raw" array can be transformed to the normal ordering using the `index_dist` indexing array. See `partial array indexing` procedure in HEDTOOLS. The table below presents an example of the two modes of ordering.

"raw"	normal	Adjusted risk
1	249	0.571031094
2	433	0.445715338
3	272	0.202977672
4	713	0.113853760
5	359	0.086924118
6	665	0.000000000

An example of the code requesting the normal array of risks. This array is arranges in the same order as as the prey `prey_spatial`:

```
call habitat_safe%predators(i)%risk_fish_group(                      &
                  prey_spatial=proto_parents%individual,             &
                  prey_length=proto_parents%individual%body_length,  &
                  risk = group_risk_array )
```

An example of the code requesting "raw" indexed array of risks along with the partial index array:

```
call this_predator%risk_fish_group(                                  &
                  prey_spatial = this%individual%location(),         &
                  prey_length = this%individual%get_length(),        &
                  is_freezing = this%individual%freeze%is_executed(), &
                  time_step_model = time_step_model_here,            &
                  risk_indexed = p_risk,                             &
                  index_dist = prey_index )
```

8.5.3.124.1.2 Notable local variables The calculations potentially involve looping over a huge array of potential prey agents, even though not all of them are at a close enough distance to the predator to have any risk. Only the agents that are within the visibility range (i.e. can be visible to the predator) count here out of the huge whole-population array. Finding the agents neighbouring to the predator benefits from partial indexing. Only a small small subarray within the input array of spatial prey agent objects is indexed and analysed. The maximum size of such a subarray is defined by the partial index size: commondata::predator_risk_group_select_index_partial (so all the work subarrays benefiting from partial indexing have such number of elements).

- **dist_index** is the partial array sorting index, see `ARRAY_INDEX()` procedure from HEDTOOLS for more details. Note that This vector comes into the the_environment::neighbours() spatial indexing procedure and must have the full size of the spatial array being indexed. So its size here is equal to the `prey_↩spatial`array size.

risk_adjusted is an array of predation risk estimates for each agent in the group. Risk for all agents that are not visible to the predator get null risk. This is the main array that is indexed exactly as the output.

- **risk_adjusted_indexed** is an array of predation risk estimates for each agent in the group. Risk for all agents that are not visible to the predator get null risk. This array that is indexed in the order of the distances between the predator and each of the nearest agents. Translation of the true index array order to this raw order requires the indexing array `dist_index`.
- **risk_agent_is_visible** is a logical flag array indicating that the i-th prey agent in the visual field of the predator is visible (i.e. within the visual range). Array limited by the maximum index size commondata::predator_risk_group_select_index_partial.
- **risk_agent_visibility** is the visibility range of each prey agent in proximity of this predator. Array limited by the maximum index size commondata::predator_risk_group_select_index_partial.

- **risk_agent_rank** is the integer rank order of each agent in the visual field of the predator with respect to the distance from the predator. Only agents visible to the predator (distance $<$ maximum visibility range) count. One potential caveat is that because the prey agents are stochastic, there can be cases when a tiny agent is the nearest to the predator, but its visibility range is very small, smaller than the distance to the predator. Such agent is not counted and has undefined (commondata::unknown) risk_agent_rank.
- **rank_visible** is the overall counter and the total number of "ranked" prey agents, i.e. those that are "visible".
- **risk_agent_baseline** is the baseline risk of predation for each of the prey agents in proximity of the predator. Array limited by the index size commondata::predator_risk_group_select_index_partial.

8.5.3.124.1.3 Checks and preparations Initialise index and rank values. Uninitialised index arrays may result in invalid memory reference in `ARRAY_INDEX` (it is not safe by design, see notes on the HEDTOOLS `array indexing` procedures.
Check if the optional `is_freezing` parameter array is provided. If not, it is assumed that the prey agents are NOT freezing.
Check optional time step parameter. If unset, use global `commondata::global_time_step_model_current`.
Check the size of the input arrays of prey agents, prey length and freezing indicztors. The sizes of all three arrays must be equal.
However, if they have different sizes, log warning, this may point to a potential bug.

8.5.3.124.1.4 Step 1 First, we get, up to the maximum order (fast *partial indexing*) of `commondata::predator_risk_group` neighbouring agents that are in proximity of this predator. Here we get **partial index** vector for the input array of objects: `dist_index`.
If the `index_dist` optional parameter is present in the list of parameters, it returns the indexing vector `dist_↩index`.

8.5.3.124.1.5 Step 2 Get a vector of agents sorted by their distances from the predator and calculate the baseline risk of predation for each of the agents using the the_environment::predator::risk_fish() method. The size of the array is limited by the maximum partial rank index `commondata::predator_risk_group_select_index_partial`, so only this number of nearest agents is taken into account. The other are too far at this moment and have null risk anyway (probably).
The boolean flag array `risk_agent_is_visible` indicating that this `i`-th agent is within the "neighbours" group is also set to TRUE.
The baseline risk of predation is the risk not taking account of the predator dilution effect by the other agent's group members in proximity.

- First, calculate the visibility (visual range) of each neighbouring prey agent.
- The risk of predation is non-zero only for those agents which are located from the predator at a distance smaller than their visibility distance, otherwise they fall outside of the visual range of the predator.
- Such agents are marked with the `risk_agent_is_visible` boolean vector value TRUE.
- Each of such visible agent is also assigned the consecutive rank, with the nearest agent having the rank 1 while the furtherst agent having the rank *N* (*N* is less than the maximum indexing rank commondata::predator_risk_group_select_index_partial). This is illustrated by the following plot.

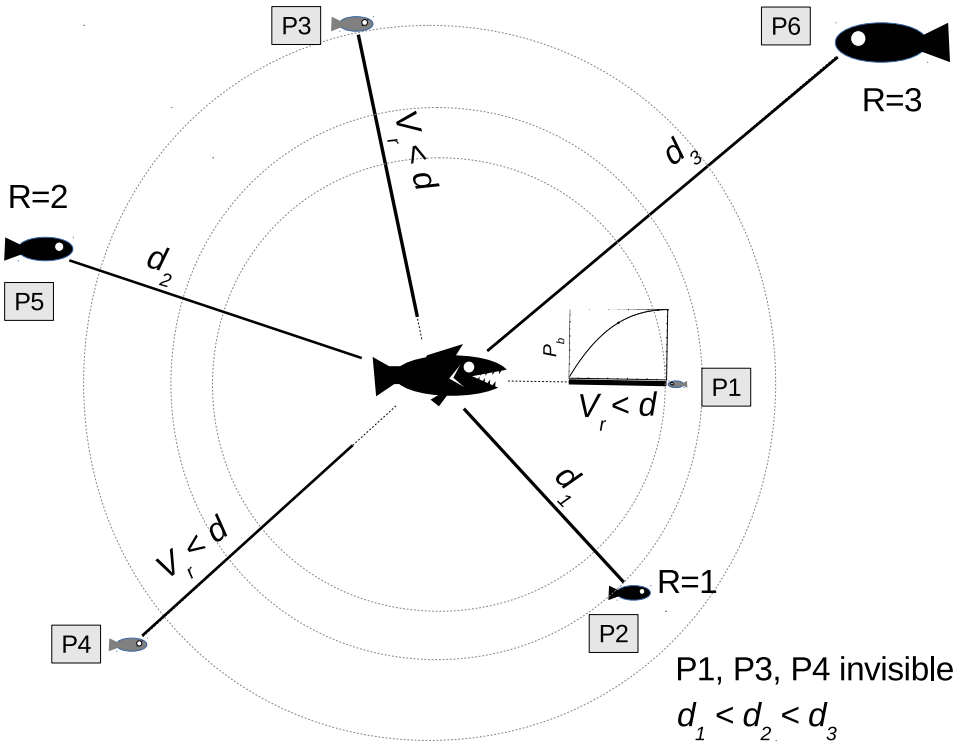


Figure 8.19 Calculation of predation risk in a group of prey agents

Here the prey agents P1, P3 and P4 are invisible to the predator (in the centre) because their visibility ranges are smaller that the distance towards the predator. P2 has the rank R=1 as it has the smallest distance to the predator among all the agents that are visible (P2, P5, P6). Notably, the agent P1 is invisible due to small body size that leads to very short visibility range (even though it is actually the closest to the predator!) and therefore zero baseline probability capture at the distance to the predator (a plot of the relationship between the distance and the baseline probability of capture is also overlaid at the agent P1).

- For all visible agents, the baseline risk of predation is calculated using the the_environment::predator::risk_fish() method. The baseline risk of predation is the risk not taking account of the predator confusion and dilution effect by the other agent's group members in proximity.

  Note

  Note that the baseline risk is zero if the distance to the predator exceeds the visual range, it is done automatically in the_environment::predator::risk_fish().

- The step 2 is finalised by checking if there were any prey agents visible to the predator. If none were visible, return from the procedure with the consequence that all adjusted risk values are equal to the initialisation value `risk_adjusted = 0.0`. This situation is also logged in the DEBUG mode.

8.5.3.124.1.6 Step 3 Given that at the step 2 we have defined which of the prey agents in the group are actually visible to the predator, their rank order with respect to the distance, and the total number of such visible agents, this final step calculates the array of the adjusted risk values.
This can be done in several ways:

- adjust_risk_nonpar_noadjust() - no specific adjustment is made for predator confusion or dilution, adjusted risk equals the baseline.
- adjust_risk_nonpar_fixed() - fixed predator confusion/dilution effect;
- adjust_risk_dilute_nofirst() - all prey except the nearest have diluted risk, on average by *1/(N-1)*, the nearest agent has the baseline risk.
- adjust_risk_dilute_all() - all prey except the closest have diluted risk, on average by *1/N*.

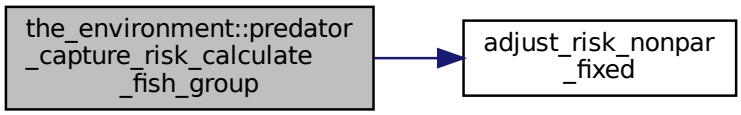
These procedires are implemented as subroutines within this main the_environment::predator_capture_risk_calcu
Calculate the raw indexed array of the risk that is intended to be returned (output) along with the partial index.
Finally, in the DEBUG mode also save debug data to CSV file. Below is an example data from one simulation using the fixed predation adjustment effect adjust_risk_nonpar_fixed().

AGENT	VISIBLE	RANK	VISIBILITY	DISTANCE	RISK_BASE	RISK_ADJ
P1	1	1	297.934	148.458	0.597635	0.597635
P2	1	2	310.591	238.575	0.375258	0.179290
P3	0	-9999	*305.785*	*306.733*	0	0
P4	0	-9999	*296.327*	*309.332*	0	0
P5	1	3	338.934	316.221	0.192204	0.034169
P6	0	-9999	*283.830*	*318.537*	0	0
P7	1	4	343.421	339.065	0	0
P8	0	-9999	311.020	366.969	0	0
P9	0	-9999	360.617	378.106	0	0
P10	0	-9999	312.113	395.021	0	0
P11	0	-9999	322.244	435.172	0	0
P12	0	-9999	335.271	442.236	0	0
P13	0	-9999	309.192	456.375	0	0
P14	0	-9999	269.335	477.448	0	0
P15	0	-9999	349.299	486.160	0	0
P16	0	-9999	273.446	501.956	0	0
P17	0	-9999	311.889	516.109	0	0
P18	0	-9999	361.142	533.081	0	0
P19	0	-9999	277.595	547.986	0	0
P20	0	-9999	267.683	561.175	0	0

Note

Note that the prey agents 3 and 4 are closer to the predator than 7, but are not visible due to small visibility range (small body sizes).

Finally, calculate the optional output arrays if the are requested as optional parameters: `risk` and `risk_↩ indexed`
Definition at line 9726 of file m_env.f90.
Here is the call graph for this function:



8.5.3.125 predator_visibility_visual_range()

```
real(srp) function the_environment::predator_visibility_visual_range (
            class(predator), intent(in) this,
            real(srp), intent(in), optional object_area,
            real(srp), intent(in), optional contrast,
            integer, intent(in), optional time_step_model )
```

Calculate the visibility range of this predator. Wrapper to the the_environment::visual_range() function. This function calculates the distance from which this predator can be seen by a visual object (e.g. the agent).

Warning

The `visual_range` procedures use meter for units, this auto-converts to cm.

Cannot implement a generic function accepting also vectors of this objects as only elemental object-bound array functions are allowed by the standard. This function cannot be elemental, so passed-object dummy argument must always be scalar.

Parameters

in	*object_area*	object_area optional area of the spatial object, m. if not provided (normally), calculated from the the_environment::predator::body_size attribute of this predator object.
in	*contrast*	contrast is optional inherent visual contrast of the predator. the default contrast of all objects is defined by the commondata::preycontrast_default parameter.
in	*time_step_model*	optional time step of the model, if absent gets the current time step as defined by the value of `commondata::global_time_step_model_current`.

Returns

The maximum distance from which this predator can be seen.

8.5.3.125.1 Implementation details **Checks.** Check optional object area, the default value, if this parameter is absent, the area is calculated from the the_environment::predator::body_size attribute of the predator object with inline conversion to m. Note that the body side area of a fish object is calculated from the body length using the commondata::length2sidearea_fish() function.
Check optional `contrast` parameter. If unset, use global `commondata::preycontrast_default`.
Check optional time step parameter. If unset, use global `commondata::global_time_step_model_current`.
Calculate ambient illumination / irradiance at the depth of this predator object at the given time step using the the_environment::spatial::illumination() method.
Return visual range for (to detect) this predator using the_environment::visual_range() wrapper function.
Definition at line 10463 of file m_env.f90.

8.5.3.126 distance_average()

```
real(srp) function the_environment::distance_average (
            class(spatial), dimension(:), intent(in) spatial_objects,
            integer, intent(in), optional sample_size )
```

Calculates the average nearest neighbour distance amongst an array of spatial objects (class) by sampling `sample_size` of them. The sample size `sample_size` is optional, if not provided set to `SAMPLE_SIZE_↩ DEFAULT=25`.

Parameters

spatial_objects	An array of spatial class objects for which we calculate the average nearest neighbour distance.
sample_size	Optional sample size for the calculation.

Returns

Returns a (sample-based) estimate of the mean nearest neighbour distance among an array of the spatial objects.

8.5.3.126.1 Notable local variables

- **SAMPLE_SIZE_DEFAULT** is the local default sample size
- **SAMPLE_SIZE_WARN** is the minimum warning sample size that results in warning logging.

MAX_ARRAY_DIMENSIONALITY: the maximum dimensionality of the input array of spatial objects for doing full non-sampling based calculations.
Maximum size of the input array of spatial objects for full element by element selection without random sampling. This maximum dimensionality can be obtained by solving this square equation:

$$M = x^2 + x,$$

where M is the maximum number of permutations. So, the maximum dimensionality of the input array of objects is

$$\frac{\sqrt{4 \cdot M + 1} + 1}{2}.$$

The value obtained is then rounded to the nearest whole integer using `nint`.

Note

Note that for the standard default `SAMPLE_SIZE_DEFAULT=25`, `MAX_ARRAY_DIMENSIONALITY` equals 6.

8.5.3.126.2 Implementation details Check the array size. Big and small arrays are treated differently.
If the array size is zero or one, there is no sense calculating average nearest neighbour distance, just return zero and log warning.
If the array size is small enough, do full element by element selection and permutation. We do not then really use the `sample_size` (or `N`) parameter.
Do full N x N permutations in such a case.

Note

Note that we here do full crossing rather than half cutting `j=i+1,array_size`. Nearest neighbour distance are **not** transitive.

In all other cases do random sampling from the input array of spatial objects and calculate sample-based average nearestneighbour distance.
Set the maximum sample size not exceeding 1/2 of all possible permutations in case the input object array is relatively large. The default value 1/2 of all possible permutations may be quite big with a huge array, but the logic behind the decision is that the requested sample size was also big.

Note

Note that we have to **convert** `integer(LONG)` to the default `integer` using the intrinsic function `int` to make it acceptable for the `TOSTR` function from HEDTOOLS. Hopefully the final `max_permutations` would not be too huge.

If the requested sample size is very small, just do a warning. The logic behind NOT changing N to, e.g. `SAMPLE_SIZE_DEFAULT` is that if a small N was requested, this must have serious grounds, e.g. if the agent is basing its decision making process on an incomplete information. So the small value is left as small as requested.
Initialise the output mean distance with zero.
Do permutations, using `PERMUTE` named do-block:

- We sample a single random spatial objects from the available array.

  Note

  Note that we have to **convert** `integer(LONG)` to the default`integer` using the intrinsic function `int` to make it acceptable for the `RAND_I` function from HEDTOOLS. Hopefully the final `array_size` would not be too huge.

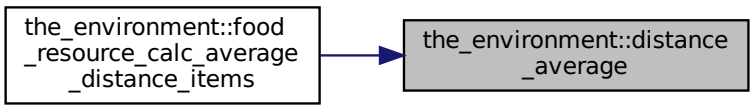
- Get an array of all other objects excluding the already sampled.

Note

There seems to be no class-safe way to do whole-array assignments of `SPATIAL` objects for high speed, e.g. using the index slice: `[ [(j, j=1,i-1)], [(j,j=i+1,array_size)] ]` as we use the type-bound function `location` and `position` to copy arays. So using the explicit old-fashioned do-loop here.

- Then we find the nearest neighbour of the i-th object out from the input array of spatial objects.
- Calculate the distance between the (randomly selected) ith object and its nearest neighbour `spatial_object_nearest_neighbour` and update the overall sum.

Finally, calculate the mean distance over all these permutations.
Definition at line 10541 of file m_env.f90.
Here is the caller graph for this function:



8.5.3.127 geo_poly2d_dist_point_to_section()

```
subroutine the_environment::geo_poly2d_dist_point_to_section (
             class(spatial), intent(in) point,
             class(spatial), intent(in) sectp1,
             class(spatial), intent(in) sectp2,
             real(srp), intent(out) min_dist,
             type(spatial), intent(out), optional point_segment )
```

Calculates the minimum distance from a the_environment::spatial class object to a line segment delimited by two the_environment::spatial class endpoints in the 2D *XY* plane (the depth coordinate is ignored). (The algorithm is partially based on `this`.)

Parameters

`in`	*point*	point is the reference point to which the distance is calculated
`in`	*sectp1*	sectp1 is the first end of the line segment.
`in`	*sectp2*	sectp2 is the second end point of the line segment.
`out`	*min_dist*	min_dist is the output minimum distance between the reference point and the line segment delimited by `sectp1` and `sectp2`.
`out`	*point_segment*	point_segment is the optional output coordinates of the nearest point **PN** on the **P1** **P2** segment returned in the form of the_environment::spatial type object. But note that the third `depth` coordinate of this object is copied from the input `point` object.

8.5.3.127.1 Implementation details

A scheme of the calculation is presented on this figure:

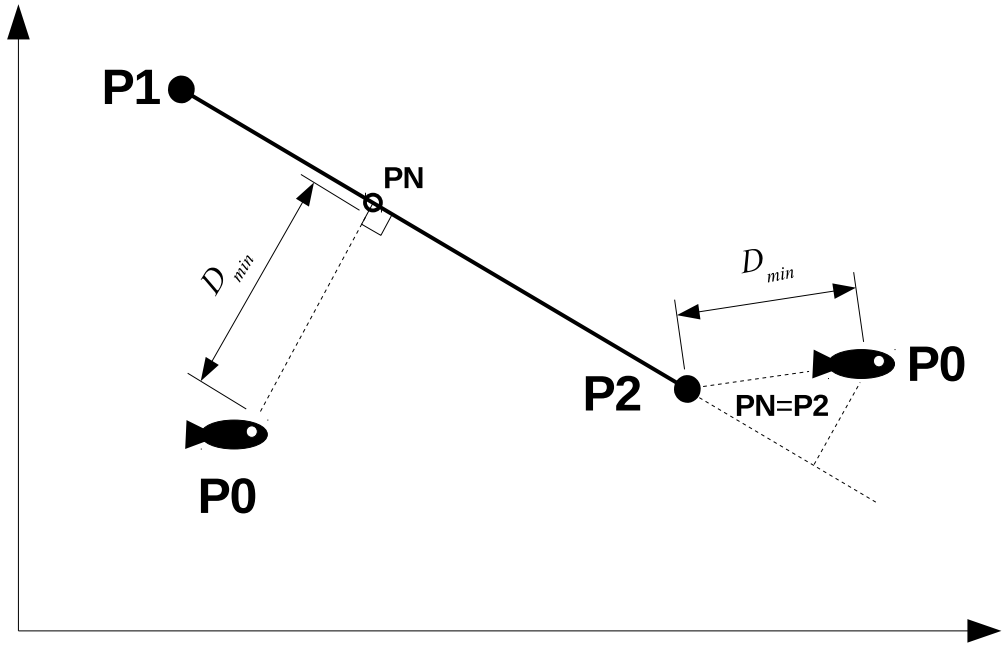


Figure 8.20 Calculation of the distance from a spatial object to a line segment

First, calculate the squared distance between the end points of the line segment **P1** and **P2** using the backend function the_environment::dist2_vector() that accepts vectors of arbitrary dimensionality: $D^2(\mathbf{p1}, \mathbf{p2})$

- if the distance between the end points is zero, the line segment actually has zero length and the distance between the spatial object **P0** and the segment is trivial to determine: $D_{min} = D(\mathbf{p_0}, \mathbf{p_1}) = D(\mathbf{p_0}, \mathbf{p_2})$.

Second, determine the reference value r (a normalised distance from **P1** to the closest point) that is calculated as follows:

$$r = \frac{|\mathbf{p}_0 - \mathbf{p}_1| \cdot |\mathbf{p}_2 - \mathbf{p}_1|}{D^2(\mathbf{p_1}, \mathbf{p_2})},$$

where $D^2(\mathbf{p_1}, \mathbf{p_2})$ is the distance between the end points **P1** and **P2**, and the numerator is the dot product of the vectors $|\mathbf{p}_0 - \mathbf{p}_1|$ and $|\mathbf{p}_2 - \mathbf{p}_1|$.

- The $r < 0$ indicates that the projection of the spatial object **P0** onto the **P1** **P2** line is located in front of the **P1** end, thus the minimum distance is calculated as

  $$D_{min} = D(\mathbf{p_0}, \mathbf{p_1}).$$

  Also, the nearest point **PN** coordinates coincide with the *X* and *Y* coordinates of **P1**.

- If $r > 1$, the projection of the spatial object **P0** on the line **P1** **P2** is behind the **P2** end, so the minimum distance is

  $$D_{min} = D(\mathbf{p_0}, \mathbf{p_2}).$$

  Also, the nearest point **PN** coordinates coincide with the *X* and *Y* coordinates of **P2**.

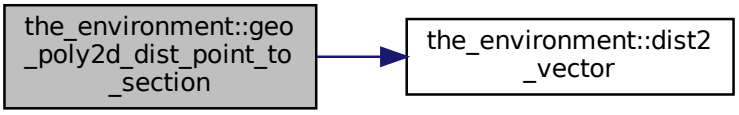
- In other cases, the spatial object **P0** projects to a specific point on the **P1** **P2** line. The distance between the spatial object **P0** and this projection is calculated as:

  $$D_{min} = \frac{|(y_2 - y_1)x_0 - (x_2 - x_2)y_0 + x_2y_1 - y_2x_1|}{D^2(\mathbf{p_1}, \mathbf{p_2})},$$

  where (x_0, y_0) are the coordinates of the spatial object **P0**, (x_1, y_1) are the coordinates of the point **P1** and (x_2, y_2) are the coordinates of the point **P2**. The nearest point **PN** coordinates are calculated as

  $$|\mathbf{p_1} + (\mathbf{p_2} - \mathbf{p_1})r|.$$

Definition at line 10769 of file m_env.f90.
Here is the call graph for this function:



8.5.3.128 geo_poly3d_dist_point_to_section()

```
subroutine the_environment::geo_poly3d_dist_point_to_section (
             class(spatial), intent(in) point,
             class(spatial), intent(in) sectp1,
             class(spatial), intent(in) sectp2,
             real(srp), intent(out) min_dist,
             type(spatial), intent(out), optional point_segment )
```

Calculates the minimum distance from a the_environment::spatial class object to a line segment delimited by two the_environment::spatial class endpoints in the 3D *XY* space. (The algorithm is partially based on `this`.)

Parameters

`in`	*point*	point is the reference point to which the distance is calculated
`in`	*sectp1*	sectp1 is the first end of the line segment.
`in`	*sectp2*	sectp2 is the second end point of the line segment.
`out`	*min_dist*	min_dist is the output minimum distance between the reference point and the line segment delimited by `sectp1` and `sectp2`.
`out`	*point_segment*	point_segment is the optional output coordinates of the nearest point **PN** on the **P1** **P2** segment returned in the form of the_environment::spatial type object.

8.5.3.128.1 Implementation details

A scheme of the calculation is presented on this figure:

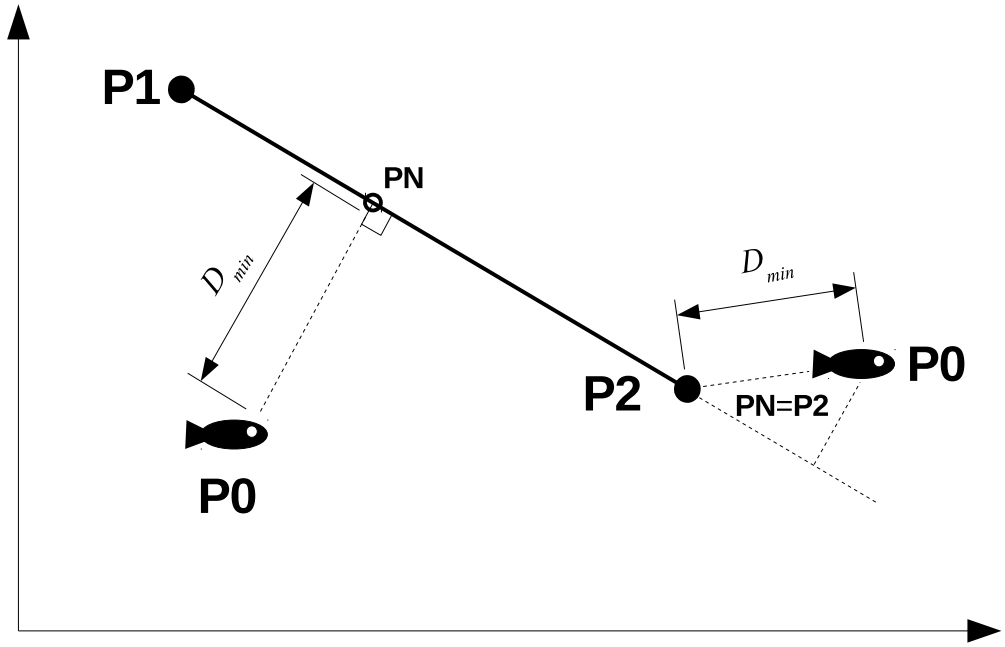


Figure 8.21 Calculation of the distance from a spatial object to a line segment

First, calculate the squared distance between the end points of the line segment **P1** and **P2** using the backend function the_environment::dist2_vector() that accepts vectors of arbitrary dimensionality: $D^2(\mathbf{p1}, \mathbf{p2})$

- if the distance between the end points is zero, the line segment actually has zero length and the distance between the spatial object **P0** and the segment is trivial to determine: $D_{min} = D(\mathbf{p_0}, \mathbf{p_1}) = D(\mathbf{p_0}, \mathbf{p_2})$.

Second, determine the reference value r (a normalised distance from **P1** to the closest point) that is calculated as follows:

$$r = \frac{|\mathbf{p}_0 - \mathbf{p}_1| \cdot |\mathbf{p}_2 - \mathbf{p}_1|}{D^2(\mathbf{p_1}, \mathbf{p_2})},$$

where $D^2(\mathbf{p_1}, \mathbf{p_2})$ is the distance between the end points **P1** and **P2**, and the numerator is the dot product of the vectors $|\mathbf{p}_0 - \mathbf{p}_1|$ and $|\mathbf{p}_2 - \mathbf{p}_1|$.

- The $r < 0$ indicates that the projection of the spatial object **P0** onto the **P1** **P2** line is located in front of the **P1** end, thus the minimum distance is calculated as

$$D_{min} = D(\mathbf{p_0}, \mathbf{p_1}).$$

- If $r > 1$, the projection of the spatial object **P0** on the line **P1** **P2** is behind the **P2** end, so the minimum distance is

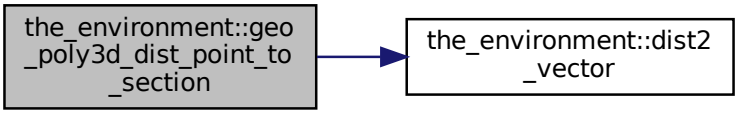
$$D_{min} = D(\mathbf{p_0}, \mathbf{p_2}).$$

- In other cases, the spatial object **P0** projects to a specific point on the **P1** **P2** line, that has these spatial coordinates:

$$\begin{cases} x_1 + r(x_2 - x_1), \\ y_1 + r(y_2 - y_1), \\ z_1 + r(z_2 - z_1) \end{cases}$$

It is then trivial to calculate the distance between the spatial object **P0** and this projection point.
Definition at line 10872 of file m_env.f90.

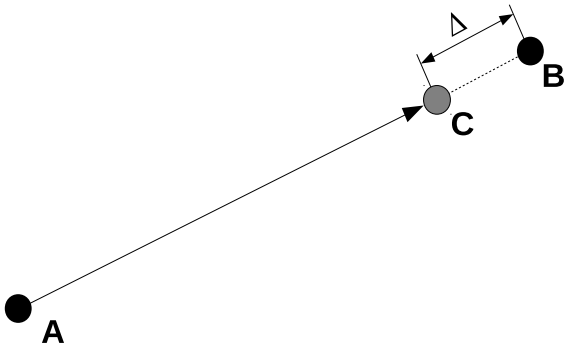
Here is the call graph for this function:



8.5.3.129 offset_dist()

```
type(spatial) function the_environment::offset_dist (
              class(spatial), intent(in) obj_a,
              class(spatial), intent(in) obj_b,
              real(srp), intent(in) offset )
```

Calculate a the_environment::spatial target with an offset.
This function calculate the coordinates of a point **C** in between two objects, a reference object **A** and a target object **B**, but at a smaller distance with specific offset Δ from the target **B**.



Parameters

in	*obj_a*	obj_a reference spatial object (**A**)
in	*obj_b*	target spatial object(**B**)
in	*obj_b*	obj_a reference spatial object (**A**)
in	*obj_b*	target spatial object(**B**)
in	*offset*	offset distance offset for the target object

Returns

Returns the coordinates of the updated target object **C** that is located at a smaller distance from **A**, by the value of the offset parameter. If the distance between **A** and **B** is smaller that the `offset` value the returned error spatial object has commondata::missing coordinates. There is no error code variable with intent(out) to keep the function pure.

The coordinate x_c of the new target **C** are defined as:

$$x_c = x_a + dist(A, B) - \Delta \cdot \frac{x_b - x_a}{dist(A, B)},$$

where x_a, x_b are the *x* coordinates of the points **A** and **B** and $dist(A, B)$ is the distance between **A** and **B**. The *y* and *depth* coordinates of the point **C** are defined in the same way.
Definition at line 10988 of file m_env.f90.

8.5.4 Variable Documentation

8.5.4.1 modname

```
character (len=*), parameter, private the_environment::modname = "(THE_ENVIRONMENT)"  [private]
```
Definition at line 25 of file m_env.f90.

8.5.4.2 dimensionality_default

```
integer, parameter, private the_environment::dimensionality_default = 3  [private]
```
Default dimensionality of the environment universe.
Definition at line 28 of file m_env.f90.

8.5.4.3 dim_environ_corners

```
integer, parameter the_environment::dim_environ_corners = 4
```
The number of corners for an environment object in the 2D *X*∗*x*∗*Y* plane.

Warning

Valid only in the simplistic box-like implementation of environment objects and should be reimplemented if the environment is set as an arbitrary polyhedron.

Definition at line 34 of file m_env.f90.

8.5.4.4 global_habitats_available

```
type(habitat), dimension(:), allocatable, public the_environment::global_habitats_available
```
A list (array) of all the the_environment::habitat objects available to the agents. This single array should encompass all the locations that the agent can potentially be in (e.g. migrate from one to another).
It is then very important that the separate habitat objects that are defined in the model are actually different data entities than the global array. If any change is made to the habitat objects after the global array was assembled, these must be synchronised with the array and vice versa.
To determine where the agent (or any other spatial object) is currently located within use the the_environment::spatial::find_environmen
method. The simplest form of assembling the global array is

```
allocate(global_habitats_available(2))
global_habitats_available = [habitat_safe, habitat_dangerous]
```

A more powerful alternative is using the the_environment::assemble() procedure:

```
call assemble(habitat_safe, habitat_dangerous, reindex=.TRUE.)
```

See the_environment::assemble() and the_environment::disassemble() procedures for more information on creating the global array of habitat objects and disassembling individual habitat objects back (updating the internal data components and arrays for each of the individual habitats.

Here is an example of the steps necessary to use joined food resource from several assembled habitats:

```
! 1. Assemble the global array of habitat objects
!    Global_Habitats_Available.
call assemble( habitat_test1, habitat_test2,                          &
               habitat_test3, habitat_test4 )
! 2. Join returns a single food resource object out of those in the
!    global array Global_Habitats_Available
joined_food_res2 = join( reindex=.true. )
! 3. Modify the joined single food resource object in some way.
!    Here it just resets the sizes of the food items for a part
!    of the data.
joined_food_res2%food( 1:size(habitat_test1%food%food) )%size = 100.0
! 4. Unjoin updates the food resources from the single global object
!    back to the global array Global_Habitats_Available.
call unjoin( joined_food_res2, reindex=.TRUE. )
! 5. To complete unjoin, the updated food habitat and resource data
!    should be transferred back to the original separate habitat objects
!    usint 'disassemble'
call disassemble( habitat_test1, habitat_test2,                       &
                  habitat_test3, habitat_test4 )
```

Note

Determining the environment object the agent is currently in can be done by the_environment::spatial::find_environment() method in this way:

```
...
        environment_limits = Global_Habitats_Available(       &
                               this_agent%find_environment(   &
                                 Global_Habitats_Available) )
...
```

Using a list of the_environment::habitat's rather than the_environment::environment's because the agent dwells in a habitat object and extended properties (e.g. habitat name) are easily available in such a case.

Warning

It is not possible to define this global variable in the commondata module because all environmental objects are defined in a higher level hierarchy module the_environment.

This array must be initialised immediately after creating the environmental objects / habitats : the_evolution::init_environment_objects().

This global array definition cannot be moved to the start of the module (for convenience), this results in the "object used before it is defined" compiler error.

Definition at line 680 of file m_env.f90.

8.6 the_evolution Module Reference

Implementation of the genetic algorithm.

Functions/Subroutines

- subroutine init_environment_objects ()

  Initialise the environmental objects. Most of the environmental objects, such as the environment, habitats etc. are kept static throughout the model running. There are, however, patterned and stochastic changes in the environment, such as diurnal variation of the illumination level.
- integer function, public preevol_steps_adaptive (generation)

  Calculate the adaptive number of time steps for the fixed fitness preevolution stage of the genetic algorithm.
- subroutine, public preevol_steps_adaptive_save_csv (csv_file_name, is_success)

  This is a diagnostic subroutine to save the number of time steps for the adaptive GA.
- subroutine generations_swap ()

  Swap generation pointers between parents and offspring.
- subroutine selection ()

  Select reproducing agents, the best commondata::ga_reproduce_pr portion of agents.

- subroutine mate_reproduce ()

  Mate, reproduce and mutate.
- subroutine, public generations_loop_ga ()

  *This procedure implements the main **Genetic Algorithm** for evolving the agents.*

Variables

- character(len= ∗), parameter, private modname = "(THE_EVOLUTION)"
- type(timer_cpu), public stopwatch_global

  Model-global stopwatch objects.
- type(timer_cpu), public stopwatch_generation
- type(timer_cpu), public stopwatch_op_current
- type(timer_cpu), public single
- type(timer_cpu), public operation
- type(habitat), public habitat_safe

  We have an environment composed of two habitats, safe and a dangerous.
- type(habitat), public habitat_dangerous
- type(population), target, public generation_one

  Here we create instances for two populations which will then serve as parents and offspring. And then we declare pointers that will point to parents and offspring.
- type(population), target, public generation_two
- type(population), pointer, public proto_parents
- type(population), pointer, public proto_offspring

8.6.1 Detailed Description

Implementation of the genetic algorithm.

8.6.2 THE_EVOLUTION module

The Genetic Algorithm is implemented here

8.6.3 Function/Subroutine Documentation

8.6.3.1 init_environment_objects()

`subroutine the_evolution::init_environment_objects [private]`

Initialise the environmental objects. Most of the environmental objects, such as the environment, habitats etc. are kept static throughout the model running. There are, however, patterned and stochastic changes in the environment, such as diurnal variation of the illumination level.

8.6.3.1.1 Build the environmental objects Build the overall environment "universe". It can be used for the whole-environment placement of objects, e.g. random walks of an agent crossing the borders between the habitats.
Build the habitats.
Define and allocate the global array of all habitats available to the agents. See the_environment::global_habitats_available for details of this global array. This is now made using the the_environment::assemble() procedure.
Allocation of the the_environment::global_habitats_available is checked. If it turns out not allocated, a critical error is signalled in the logger and the program calls commondata::system_halt().

8.6.3.1.2 Save initial diagnostic data Output the number of the habitats in the global array the_environment::global_habitats_avail and their labels into the logger.
Certain data are also saved. Their names start from the `init_` prefix.

- Save initial food data (uniform distribution as built at init). Note that the distribution of the food items can change at each time step due to vertical migration of the food items and their local random Gaussian movements.

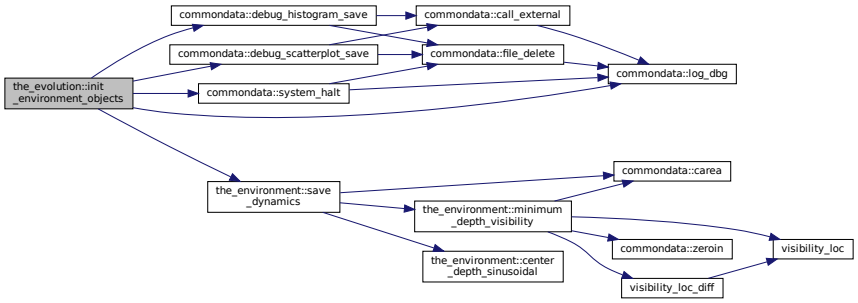
Save predators' data.

- Save the basic data on the dynamics of illumination, food items and visibility across the life span of the agents.

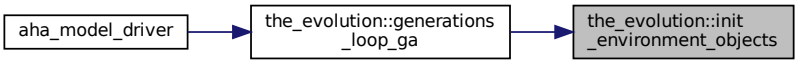
8.6.3.1.2.1 Save plots If the plotting is enabled (see commondata::is_plotting), some plots of the initialisation data are also saved.

- Save debug scatterplots of food items distribution within in the habitats.
- Save debug scatterplots of predators distribution in the habitats.
- Save histograms of food item sizes.

Definition at line 62 of file m_evolut.f90.
Here is the call graph for this function:



Here is the caller graph for this function:



8.6.3.2 preevol_steps_adaptive()

```
integer function, public the_evolution::preevol_steps_adaptive (
            integer, intent(in), optional generation )
```

Calculate the adaptive number of time steps for the fixed fitness preevolution stage of the genetic algorithm.
The number of time steps in the fixed-fitness pre-evolution genetic algorithm is calculated using an adaptive algorithm. Briefly, the number of time steps (total lifespan) at the early stages of evolution (the first generations) is very short and increases as the evolution proceeds towards the maximum set by commondata::preevol_tsteps.

Note

The time steps data generated by this function for each GA generation are saved in CSV file by the_evolution::preevol_steps_adaptive_save_csv().

Parameters

in	*generation*	generation optional current generation number, if not provided, set to commondata::global_generation_number_current.

Returns

The number of lifecycle time steps at the specific generation.

8.6.3.2.1 Implementation notes The number of time steps in this fixed fitness pre-evol adaptive GA algorithm is calculated based on a linear interpolation from a grid defined by the two arrays:

- `STEPS_ABSCISSA` – grid abscissa, from the first generation to the total number of generations commondata::generations.

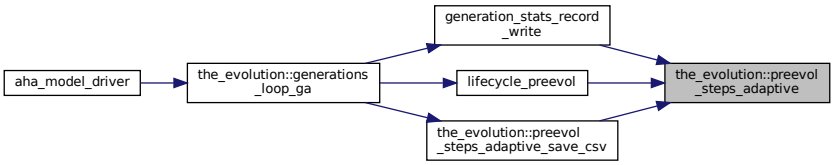
`STEPS_ORDINATE` – grid ordinate, ranging from the number of time steps in one diel cycle to the total number of time steps in the fixed fitness pre-evolution stage commondata::preevol_tsteps.
However, for debugging purposes, evolution time steps can be set to a specific fixed value. This value is set by commondata::preevol_tsteps_force_debug integer parameter and for this fixed value to be forced, commondata::preevol_tsteps_force_debug_enabled must be TRUE.
Then, the total (adaptive) number of time steps is determined by the integer lower limit (floor) of the linear interpolation DDPINTERPOL() procedure, with further limitation that its result value must be within the range of $[t,T]$, where t is the length of a single diel cycle, T is the number of time steps in the pre-evolution stage.

Note

Plotting commands:

- `htintrpl.exe [1 50 75 101] [0 0.8 0.95 1] [1] [nonlinear]`

Definition at line 211 of file m_evolut.f90.
Here is the caller graph for this function:



8.6.3.3 preevol_steps_adaptive_save_csv()

```
subroutine, public the_evolution::preevol_steps_adaptive_save_csv (
             character(len=*), intent(in) csv_file_name,
             logical, intent(out), optional is_success )
```

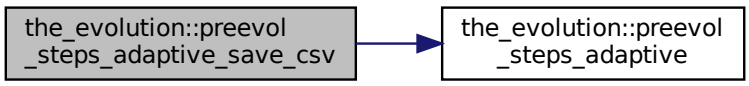
This is a diagnostic subroutine to save the number of time steps for the adaptive GA.

Parameters

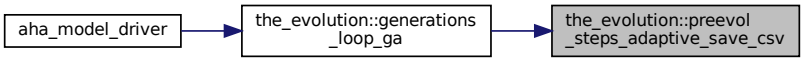
in	*csv_file_name*	csv_file_name the name of the CSV file to save the arrays.
out	*is_success*	is_success Flag showing that data save was successful (if TRUE).

Definition at line 289 of file m_evolut.f90.

Here is the call graph for this function:

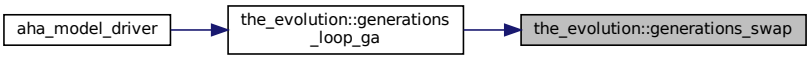


Here is the caller graph for this function:



8.6.3.4 generations_swap()

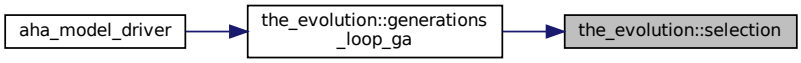
`subroutine the_evolution::generations_swap [private]`
Swap generation pointers between parents and offspring.
Definition at line 318 of file m_evolut.f90.
Here is the caller graph for this function:



8.6.3.5 selection()

`subroutine the_evolution::selection [private]`
Select reproducing agents, the best commondata::ga_reproduce_pr portion of agents.
The best (sorted) parents are copied to the offspring population object. Note that the number of such reproducing parents is determined by the the_population::population::ga_reproduce_max() method.
Old fixed proportion implementation:
`proto_offspring(:ga_reproduce_n) = proto_parents(:ga_reproduce_n)`
The best parents (elite group) are then re-initialised from the genome for the next generation using the_individual::individual_agent::init() method.
Definition at line 333 of file m_evolut.f90.

Here is the caller graph for this function:



8.6.3.6 mate_reproduce()

`subroutine the_evolution::mate_reproduce [private]`

Mate, reproduce and mutate.
Calculate adaptive mutation rate
Loop through all the non-elite population members. These individuals are created from the genomes of the elite group. The non-elite individuals are from commondata::ga_reproduce_n+1 to commondata::popsize.

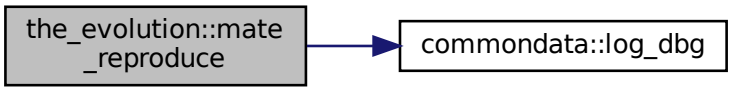
- If chromosomes are not allocated, this means it is a new individual. We have to initialise it – now as random. The same is true for all individuals that the_genome::individual::genome::is_dead().
- Two agents are randomly chosen from the population. They become the mother and the father of new `proto_offspring` agents. The mother and the father exchange their genetic material using the the_genome::individual_genome::recombine_random() method. Note that the mother must be the_genome::individual_genome::is_female() and the father, the_genome::individual_genome::is_male().
- Once the genome of the offspring is created from recombination data, the offspring are subjected to random mutation using the the_genome::individual_genome::mutate() backend.
- After this, the whole agent is initialised using he constructor the_genome::individual_agent::init(), but without random initialisation of the genome, the latter is based on the recombination data from the parents.

Finally, loop through the elite group and introduce random mutations there too with the_genome::individual_genome::mutate().

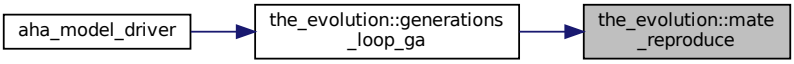
Note

This is disabled (elitism).

Definition at line 364 of file m_evolut.f90.
Here is the call graph for this function:



Here is the caller graph for this function:



8.6.3.7 generations_loop_ga()

`subroutine, public the_evolution::generations_loop_ga`
This procedure implements the main **Genetic Algorithm** for evolving the agents.

8.6.3.7.0.1 Objects for the GA

- `energy_mean_gen1_birth_mort` – average value of the birth energy reserves, for forced selective birth mortality. See the_population::population::mortality_birth().

`energy_sd_gen1_birth_mort` – standard deviationof the birth energy reserves, for forced selective birth mortality. See the_population::population::mortality_birth().

8.6.3.7.0.2 Objects for generation-wise statistics file The definitions below are for the objects that are used to write generation-wise statistics in the generation_stats_record_write() sub-procedure.

- `csv_file_generstats`: `CSV_IO` File handle object for generation-wise statistics. See `CSV_IO` module for details.

  Note

  Note that file_io::file_handle object is not used here because it breaks Intel Fortran compiler: as the file unit somehow gets lost in generation_stats_record_write(). GNU gfortran has no issues here.

`file_stats_gener_record`: Record for the generation-wise statistics file.

- `FILE_STATS_GENER_COLS`: an array of column names for the generation-wise statistics file.
- `FILE_STATS_RECORD_LEN`: The maximum length of the CSV record assuming the maximum length of a single field is commondata::label_length; the number of fields is equal to the size of the columns array `FILE_STATS_GENER_COLS`.

8.6.3.7.0.3 Parameters for the GA stopping rule Parameters determining the **stopping rule** for the fixed fitness genetic algorithm. These are based on the values obtained in the first generation. If in any succeeding generation, they fall below the first generation values, evolution is considered unsuccessful and the main GA loop stops. The number of alive agents at the first random generation.

- Evolution should stop with unsuccessful status if the number of alive agents falls below this value.

The number of agents that have increased their body mass at the first random generation.

- Evolution should stop with unsuccessful status if the number of alive agents falls below this value.

8.6.4 Preliminary steps

`Global_Generation_Number_Current` is the global generation number. It is first initialised to **1**.
commondata::Global_Rescale_Maximum_Motivation is the global maximum motivation value, it is fixed at the start of the simulation to an arbitrary high value but is automatically updated from the maximum motivation value across all agents after each time step.
The stopping rule parameters based on the first generation values are initialised to some values allowing the first generation to occur safely, i.e. with sufficiently large number of randomly created pre-optimal agents.

- If the number of alive agents is smaller than this minimum number, GA stops: some parameters must be tweaked.

The number of agents growing is set to a large negative value commondata::unknown,so initial zero is always larger, so evolution is allowed to start.

8.6.4.1 Initialise the environment

All environmental objects are initialised with init_environment_objects().

8.6.4.2 Initialise base agent population objects

New populations of agents are now built and initialised: (a) `generation_one`, (b) `generation_two` These population objects serve as targets for two pointer objects: (a) `proto_parents`, (b) `proto_offspring`.

- Initialise the whole `generation_one` of the agents, commondata::popsize is the size of the population.
- Also initialise the `generation_two`, that will then take parents' values.

Calculate initial fitness of the agents in the `generation_one` for the pre-evolution phase. At this stage fitness is equal to the maximum value (note that fitness is actually a reverse of fitness) and is not very interesting.
Place all the agents that have been initialised to random spatial positions in the safe habitat (`habitat_safe`), they have just the uniformly distributed spatial positions at start.

Note

Note that the initial vertical position and distribution of the agents depends on these parameters:

- commondata::init_agents_depth_is_fixed
- commondata::init_agents_depth_is_gauss

See the_population::individ_posit_in_environ_uniform() for details.

8.6.4.3 Transfer pointers: parents and offspring populations

Allocate the first `proto_parents` and `proto_offspring` population objects, they are pointers to `generation_one` and `generation_two` target objects.
Calculate statistical parameters of the initial generation for selective birth mortality. See the_population::population::mortality_birth().
These values are then logged.

8.6.4.4 Save diagnostics data

Save initialisation data in the debug mode.

- Saving histograms of agents' body length.
- Saving histograms of agents' body mass.
- Saving histograms of agents' energy.
- Saving histograms of agents' smr.

SAVE_DATA_INIT block: The random initialisation individual data for the whole parent population are saved to csv files:

- Individual agent's data, file `init_agents_`;
- Initial genome data, file `init_genome_`.
- The number of time steps in the adaptive GA

The generation wise statistics file `generations_` is opened for writing ...
... and the first row of column names `FILE_STATS_GENER_COLS` is written.

Note

Note that the main body of the statistical data is processed in the sub-procedure generation_stats_record_write().

The average distance between the food items is reported to the log. The average distance between the food items is good to know, e.g. to compare it with the agent's random walk step size.

8.6.5 Pre-evolution stage

Pre-evolution stage involves the Genetic Algorithm that is based on selection of agents based on an explicit global fitness. It aims to produce a population of agents that can stably sustain for the whole commondata::lifespan

8.6.5.1 GENERATIONS_PREEVOL: The main loop of (pre-) evolution

At this stage the main loop of generations evolving is started. The conditions for **continuing** the main evolution loop are as follows:

8.6.5.1.0.1 Check stop file The commondata::stop_file file is checked upon each generation. If this stop file exists, the simulation cycle is terminated at this stage and the program then exits from the generation loop.

8.6.5.1.0.2 Diagnostics Stopwatch object for calculating time since generation start is initialised.
Initially, place all the agents in the `proto_parents` population randomly uniformly in the safe habitat (`habitat_safe`). However, note that the initial vertical position and distribution of the agents depends on these parameters:

- commondata::init_agents_depth_is_fixed
- commondata::init_agents_depth_is_gauss

See the_population::individ_posit_in_environ_uniform() method for details.
Initialise the global generation-wise counter of the number of agents that die as a consequence of predation the_population::global_ind_n_eaten_by_predators, as opposed to starvation.
If it is **not** the first generation, replenish all food items (i.e. for all habitats), they are restored to the "available" (non-eaten) state. Two methods can be used here:

- the_environment::food_resource::make() – re-initialise food items from scratch.
- the_environment::food_resource::replenish() – reuse food items as initialised in init_environment_objects()

The global habitat array the_environment::global_habitats_available is then updated by the_environment::assemble() procedure.
If it is the first generation, it does not make sense doing this as the environment has been already fully initialised in the init_environment_objects() procedure.

8.6.5.1.1 lifecycle_preevol for proto_parents Start the loop of the life cycle of all agents of the `proto_↩ parents`. It includes commondata::preevol_tsteps time steps. (Note that commondata::preevol_tsteps is less than commondata::lifespan). This is implemented in the lifecycle_preevol() procedure.
Calculate the number of agents alive and agents growing. These values are used later, including as a criterion of GA deterioration.
Report these values in the logger.
After the agents went through their life cycle, their fitness is processed.

- Fitness of all `proto_parents` agents is recalculated following their performance in the full lifecycle.

The agents `proto_parents` are sorted by fitness.
If this is the first generation, determine the GA deterioration stopping parameters, evolution "failure"
These Generation one parameters are also reported to the logger.

- **SAVE_DATA_INDS_GENERATION block**: The individual statistical data for the whole `proto_parents` population are saved using the the_population::population class bound `save_` methods:
- Individual agent's data, file `agents_`
- The genome data, file `genome_`
- Memory stacks data, file `memory_`.
- Movement history data, file `movements_`.
- Behaviour history data, file `behaviours_`.
- **SAVE_DATA_FOOD_POST**: The food resources data for all the habitats are saved using the the_environment::food_resource::save_csv() method.
- Generation-wise statistics are calculated and saved in the CSV file. This is implemented in the generation_stats_record_write() subprocedure.

- Check if the unsuccessful evolution criterion is met. If yes, terminate the GA.
  - The number of agents that are alive exceeds that in the first generation: there must be improvement (at least in debug).
  - The number of agents that have grown exceeds that in the first generation.
- If this is the first generation, terminate GA if the number of agents alive $<$ 1/100 of the commpndata::popsize or if no agents are growing.

8.6.5.1.2 Selection Select reproducing minority: the_evolution::selection()

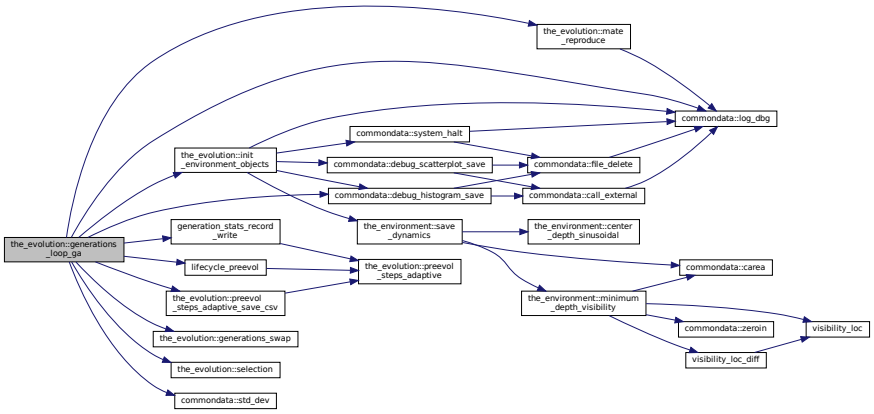
8.6.5.1.3 Exchange of the genetic material A minority of the best parents produces the proto_offspring population object: the_evolution::mate_reproduce().
Reset individual IDs of proto_offspring

8.6.5.1.4 Finalise the generation cycle: swap pointers Swap populations: `proto_offspring` are now `proto_parents`: the_evolution::generations_swap().

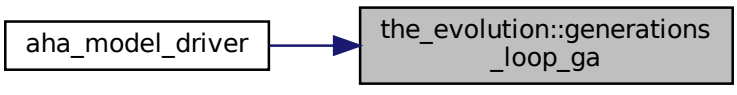
8.6.5.1.5 End of the generations loop Finally, the global generation counter commondata::global_generation_number_current is incremented by one.
After all generations were done, the CSV file `csv_file_generstats` that saves generation-wise statistics is closed.

8.6.6 System terminates

Finally, the concluding procedure commondata::system_halt() is called for the normal termination of the model.
Definition at line 445 of file m_evolut.f90.
Here is the call graph for this function:



Here is the caller graph for this function:



8.6.7 Variable Documentation

8.6.7.1 modname

```
character (len=*), parameter, private the_evolution::modname = "(THE_EVOLUTION)"  [private]
```
Definition at line 35 of file m_evolut.f90.

8.6.7.2 stopwatch_global

```
type(timer_cpu), public the_evolution::stopwatch_global
```
Model-global stopwatch objects.

Note

Use the keyword `TIMER:` (LTAG_TIMER) for logging, e.g. `call LOG_MSG( LTAG_TIMER // stopwatch_op_currentshow() )`

Definition at line 40 of file m_evolut.f90.

8.6.7.3 stopwatch_generation

```
type(timer_cpu), public the_evolution::stopwatch_generation
```
Definition at line 40 of file m_evolut.f90.

8.6.7.4 stopwatch_op_current

```
type(timer_cpu), public the_evolution::stopwatch_op_current
```
Definition at line 40 of file m_evolut.f90.

8.6.7.5 single

```
type(timer_cpu), public the_evolution::single
```
Definition at line 40 of file m_evolut.f90.

8.6.7.6 operation

```
type(timer_cpu), public the_evolution::operation
```
Definition at line 40 of file m_evolut.f90.

8.6.7.7 habitat_safe

```
type(habitat), public the_evolution::habitat_safe
```
We have an environment composed of two habitats, safe and a dangerous.
Definition at line 45 of file m_evolut.f90.

8.6.7.8 habitat_dangerous

```
type(habitat), public the_evolution::habitat_dangerous
```
Definition at line 45 of file m_evolut.f90.

8.6.7.9 generation_one

`type(population), target, public the_evolution::generation_one`
Here we create instances for two populations which will then serve as parents and offspring. And then we declare pointers that will point to parents and offspring.
Definition at line 50 of file m_evolut.f90.

8.6.7.10 generation_two

`type(population), target, public the_evolution::generation_two`
Definition at line 51 of file m_evolut.f90.

8.6.7.11 proto_parents

`type(population), pointer, public the_evolution::proto_parents`
Definition at line 52 of file m_evolut.f90.

8.6.7.12 proto_offspring

`type(population), pointer, public the_evolution::proto_offspring`
Definition at line 53 of file m_evolut.f90.

8.7 the_genome Module Reference

Definition the genetic architecture of the agent.

Data Types

- type gene

  This describes an individual gene object. See the genome structure for as general description and gene for details.
- type chromosome

  This type describes the chromosome object. Chromosome consists of an array of alleles and a descriptive string label. See "the genome structure" for as general description and "chromosome" for details.
- type individual_genome

  This type describes parameters of the individual agent's genome The genome is an array of allocatable the_genome::chromosome objects, different kinds of agents may have different genomes with different number of chromosomes. See "the genome structure" for as general description and "genome" for details.

Functions/Subroutines

- elemental subroutine chromosome_sort_rank_id (this)

  Sort GENE objects within the CHROMOSOME by their `rank_id`. The two subroutines `qsort` and `qs_partition_rank_id` are a variant of the recursive quick-sort algorithm adapted for sorting integer components of the the `CHROMOSOME` object.
- subroutine allele_init_random (this)

  *Initialises allele with a random integer. Note that we do **not** set the labels for the alleles here during the random initialisation.*
- elemental subroutine allele_create_zero (this)

  Create allele with zero value. We don't set labels for alleles here.
- subroutine allele_label_init_random (this)

  The (pair of) alleles here are assigned random string labels Not sure if that is necessary for any application though.
- elemental subroutine allele_label_set (this, label)

  Set labels for the alleles. The subroutine parameter is array of labels.

- elemental character(len=label_length) function allele_label_get (this)

  Get the i-th allele label.
- elemental subroutine allele_value_set (this, set_value, nr)

  Set a single value of the allele additive component.
- pure subroutine alleles_value_vector_set (this, values)

  Set values of the alleles as a vector, i.e. sets the whole gene values.
- elemental integer function allele_value_get (this, nr)

  Get the value of the allele.
- pure subroutine allele_values_vector_get (this, values)

  Get the vector of all values of the alleles, i.e. gets the gene values.
- elemental subroutine allele_rank_id_set (this, value_id)
- subroutine allele_mutate_random (this, prob)

  Introduce a random point mutation to a random allele component.
- subroutine allele_mutate_random_batch (this, prob)

  Introduce a random mutation of the whole set of additive allele components.
- subroutine chromosome_init_allocate_random (this, length, label)

  This subroutine initialises the chromosome with, and allocates, random alleles, sets one of them randomly dominant and optionally defines the chromosome label.
- subroutine chromosome_create_allocate_zero (this, length, label)

  Init a new chromosome, zero, non-random.
- elemental subroutine chromosome_recalculate_rank_ids (this)

  This subroutine recalculates rank_id indices for consecutive gene objects within the chromosome. This may be necessary after reordering by random relocation mutation.
- subroutine chromosome_mutate_relocate_swap_random (this, prob)

  *Mutate within the same chromosome, relocate a gene (unit of alleles) to a different random position within the same chromosome, the misplaced gene moves to the relocated gene position, so they are just **swap**.*
- subroutine chromosome_mutate_relocate_shift_random (this, prob)

  *Mutate within the same chromosome, relocate a gene (unit of alleles) to a different random position within the same chromosome, **shifting** all other genes within the chromosome down one position. This works as follows: first, we randomly determine the gene to relocate, assign it a new random rank_id. Then re-sort the chromosome according to the new ranks with `qsort` with the `qs_partition_rank_id` backend.*
- subroutine genome_init_random (this, label)

  Initialise the genome at random, and set sex as determined by the sex determination locus.
- subroutine genome_create_zero (this)

  Create a new empty genome, and set sex as determined by the sex determination locus. Genome values are from parents using inherit functions.
- subroutine genome_label_set (this, label)

  Label genome. If label is not provided, make a random string.
- elemental character(len=label_length) function genome_label_get (this)

  Accessor function to get the genome label. The label is a kind of a (random) text string name of the genome and the individual agent.
- subroutine genome_sex_determine_init (this)

  Sex determination initialisation subroutine.
- elemental logical function genome_get_sex_is_male (this)

  Get the logical sex ID of the genome object component.
- elemental logical function genome_get_sex_is_female (this)

  Get the logical sex ID of the genome object component.
- elemental character(len=label_length) function genome_get_sex_label (this)

  Get the descriptive sex label: male or female.
- elemental subroutine genome_individual_set_alive (this)

  *Set the individual to be **alive**, normally this function is used after init or birth.*
- elemental subroutine genome_individual_set_dead (this)

*Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function.*

- elemental logical function genome_individual_check_alive (this)

  *Check if the individual is **alive**.*
- elemental logical function genome_individual_check_dead (this)

  *Check if the individual is **dead** (the opposite of `is_alive`).*
- subroutine genome_individual_recombine_homol_full_rand_alleles (this, mother, father, exchange_ratio)

  Internal genetic recombination backend, exchange individual alleles between homologous chromosomes in mother and father genomes to form the `this` (offspring) genome. Fully random recombination.
- subroutine genome_individual_recombine_homol_part_rand_alleles (this, mother, father, exchange_ratio)

  Internal genetic recombination backend, exchange individual alleles between homologous chromosomes in mother and father genomes to form the `this` (offspring) genome. Partially random recombination, identical across the homologous chromosomes.
- subroutine genome_individual_crossover_homol_fix (this, mother, father, pattern_matrix)

  Internal fixed genetic crossover backend, exchange blocks of alleles between homologous chromosomes in mother and father genomes to form the `this` (offspring) genome.
- subroutine genome_mutate_wrapper (this, p_point, p_set, p_swap, p_shift)

  Perform a probabilistic random mutation(s) on the individual genome. This is a high level wrapper to build mutations from various components.

Neuronal response functions

There are two separate functions that produce a trait value from the genotype. The procedure `trait_init_genotype_gamma2gene` does modify the agent (`this`, intent[inout]) as it sets the label. On the other hand, a similar procedure `the_genome::trait_set_genotype_gamma2gene()` does not affect the agent, it has the intent [in].

- subroutine trait_init_genotype_gamma2gene (this, this_trait, g_p_matrix, init_val, gerror_cv, label)

  *Initialise an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy.*
- subroutine trait_set_genotype_gamma2gene (this, this_trait, g_p_matrix, init_val, gerror_cv)

  *Set an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy.*

The genotype to phenotype functions based on fixed linear

transformation.

- subroutine trait_init_linear_sum_additive_comps_2genes_r (this, this_trait, g_p_matrix, phenotype_min, phenotype_max, label)

  *Initialise an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy.*
- subroutine trait_set_linear_sum_additive_comps_2genes_r (this, this_trait, g_p_matrix, phenotype_min, phenotype_max)

  *Set an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy.*

Variables

- character(len= $*$), parameter, private modname = "(THE_GENOME)"

8.7.1 Detailed Description

Definition the genetic architecture of the agent.

8.7.2 THE_BODY module

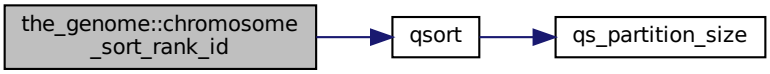
This module defines the genetic architecture objects of the agent. See The genome structure for an overview.

8.7.3 Function/Subroutine Documentation

8.7.3.1 chromosome_sort_rank_id()

```
elemental subroutine the_genome::chromosome_sort_rank_id (
            class(chromosome), intent(inout) this )
```

Sort GENE objects within the CHROMOSOME by their `rank_id`. The two subroutines `qsort` and `qs_partition_rank_id` are a variant of the recursive quick-sort algorithm adapted for sorting integer components of the the `CHROMOSOME` object.
Definition at line 323 of file m_genome.f90.
Here is the call graph for this function:



8.7.3.2 allele_init_random()

```
subroutine the_genome::allele_init_random (
            class(gene), intent(inout) this )
```

Initialises allele with a random integer. Note that we do **not** set the labels for the alleles here during the random initialisation.
The allele components are initialised by a random integers within the range `ALLELERANGE_MIN` and `ALLELERANGE_MAX` parameetr values.
Definition at line 410 of file m_genome.f90.

8.7.3.3 allele_create_zero()

```
elemental subroutine the_genome::allele_create_zero (
            class(gene), intent(inout) this )
```

Create allele with zero value. We don't set labels for alleles here.

Note

Note that there is no need to allocate `allele_value` array as it has fixed shape.

Definition at line 424 of file m_genome.f90.

8.7.3.4 allele_label_init_random()

```
subroutine the_genome::allele_label_init_random (
            class(gene), intent(inout) this )
```

The (pair of) alleles here are assigned random string labels Not sure if that is necessary for any application though.
Definition at line 436 of file m_genome.f90.

8.7.3.5 allele_label_set()

```
elemental subroutine the_genome::allele_label_set (
            class(gene), intent(inout) this,
            character(len=*), intent(in) label )
```

Set labels for the alleles. The subroutine parameter is array of labels.

Parameters

in	*label*	label, provides an array of labels to set for the allele.

Definition at line 445 of file m_genome.f90.

8.7.3.6 allele_label_get()

```
elemental character(len=label_length) function the_genome::allele_label_get (
            class(gene), intent(in) this )
```

Get the i-th allele label.

Returns

Returns the label of the allele.

Definition at line 456 of file m_genome.f90.

8.7.3.7 allele_value_set()

```
elemental subroutine the_genome::allele_value_set (
            class(gene), intent(inout) this,
            integer, intent(in) set_value,
            integer, intent(in) nr )
```

Set a single value of the allele additive component.

Parameters

in	*set_value*	value, provides the value to set for the allele and the allele number.
in	*nr*	number, provides the number of the allele component to set.

Definition at line 467 of file m_genome.f90.

8.7.3.8 alleles_value_vector_set()

```
pure subroutine the_genome::alleles_value_vector_set (
            class(gene), intent(inout) this,
            integer, dimension(additive_comps), intent(in) values )
```

Set values of the alleles as a vector, i.e. sets the whole gene values.

Parameters

in	*values*	values, provides vector of values to set for the alleles.

Definition at line 482 of file m_genome.f90.

8.7.3.9 allele_value_get()

```
elemental integer function the_genome::allele_value_get (
            class(gene), intent(in) this,
            integer, intent(in) nr )
```

Get the value of the allele.

Parameters

in	*nr*	number, provides the number of the allele component to set.

Returns

Returns the value of the nr's allele.

Definition at line 493 of file m_genome.f90.

8.7.3.10 allele_values_vector_get()

```
pure subroutine the_genome::allele_values_vector_get (
            class(gene), intent(in) this,
            integer, dimension(additive_comps), intent(out) values )
```

Get the vector of all values of the alleles, i.e. gets the gene values.

Parameters

out	*values*	values, Gets the vector of the values for the alleles.

Definition at line 508 of file m_genome.f90.

8.7.3.11 allele_rank_id_set()

```
elemental subroutine the_genome::allele_rank_id_set (
            class(gene), intent(inout) this,
            integer, intent(in) value_id )
```

Parameters

in	*value_id*	rank_id, set this value to the allele rank_id.

Definition at line 518 of file m_genome.f90.

8.7.3.12 allele_mutate_random()

```
subroutine the_genome::allele_mutate_random (
            class(gene), intent(inout) this,
            real(srp), intent(in), optional prob )
```

Introduce a random point mutation to a random allele component.

Parameters

in	*prob*	prob optional probability of mutation, if absent, the default value commondata::mutationrate_point is used.

8.7.3.12.1 Implementation details Do mutate if a random value is smaller than the commondata::mutationrate parameter constant.
First, determine which of the alleles components gets mutation.
Second, change the value of this allele component to an random integer.
Definition at line 530 of file m_genome.f90.

8.7.3.13 allele_mutate_random_batch()

```
subroutine the_genome::allele_mutate_random_batch (
            class(gene), intent(inout) this,
            real(srp), intent(in), optional prob )
```

Introduce a random mutation of the whole set of additive allele components.

Parameters

in	*prob*	prob optional probability of mutation, if absent, the default value commondata::mutationrate_batch is used.

8.7.3.13.1 Implementation details This mutation just re-init the whole allele set as random.
Definition at line 562 of file m_genome.f90.

8.7.3.14 chromosome_init_allocate_random()

```
subroutine the_genome::chromosome_init_allocate_random (
            class(chromosome), intent(inout) this,
            integer, intent(in) length,
            character(len=*), intent(in), optional label )
```

This subroutine initialises the chromosome with, and allocates, random alleles, sets one of them randomly dominant and optionally defines the chromosome label.

Parameters

in	*length,sets*	the length of the chromosome object, N of alleles.
in	*label,sets*	the label for the chromosome object, optional.

8.7.3.14.1 Implementation details We set the chromosome label if such a parameter is provided, or a random string if not.
First, set the chromosome length using the procedure parameter `length`.
Then, allocate the array of the allele objects with this length.
Initialise all the alleles within this chromosome.
Specifically, initialise the allele.
Set initial rank_id ID of the allele.
Finally, set the label for the alleles within this chromosome. Labels can be set random using this function (disabled):
`call this%allele(i)%label_random()`
But in this implementation we construct the label for the allele from the *chromosome label* and the *allele number*;
Long chromosome labels are trimmed at right to fit the allele number.
Definition at line 596 of file m_genome.f90.

8.7.3.15 chromosome_create_allocate_zero()

```
subroutine the_genome::chromosome_create_allocate_zero (
            class(chromosome), intent(inout) this,
            integer, intent(in) length,
            character(len=*), intent(in), optional label )
```

Init a new chromosome, zero, non-random.

8.7.3.15.1 Implementation details Set the chromosome label if provided as parameter, or random string if not.
First, set the chromosome length using the procedure parameter `length`.
Then, we allocate the array of the allele objects with this length.
Initialise all the alleles within this chromosome.

Note

Parallel `do concurrent` construct is used here.

Definition at line 647 of file m_genome.f90.

8.7.3.16 chromosome_recalculate_rank_ids()

```
elemental subroutine the_genome::chromosome_recalculate_rank_ids (
            class(chromosome), intent(inout) this )
```

This subroutine recalculates rank_id indices for consecutive gene objects within the chromosome. This may be necessary after reordering by random relocation mutation.
Definition at line 697 of file m_genome.f90.

8.7.3.17 chromosome_mutate_relocate_swap_random()

```
subroutine the_genome::chromosome_mutate_relocate_swap_random (
            class(chromosome), intent(inout) this,
            real(srp), intent(in), optional prob )
```

Mutate within the same chromosome, relocate a gene (unit of alleles) to a different random position within the same chromosome, the misplaced gene moves to the relocated gene position, so they are just **swap**.

Parameters

in	*prob*	prob optional probability of mutation, if absent, the default value commondata::relocation_swap_rate is used.

8.7.3.17.1 Implementation details Do mutate if a random value is smaller than the commondata::relocation_swap_rate parameter constant value.
If yes, randomly determine the gene (`gene_move`) that initiates the mutation move within the chromosome.
Randomly determine the gene that will be swapped with the `gene_move`.
Then, cycle through the alleles and select new random allele if it happens to coincide with `gene_move`.
After this, do the gene swap, and gene rank_id ID swap.
Definition at line 714 of file m_genome.f90.

8.7.3.18 chromosome_mutate_relocate_shift_random()

```
subroutine the_genome::chromosome_mutate_relocate_shift_random (
            class(chromosome), intent(inout) this,
            real(srp), intent(in), optional prob )
```

Mutate within the same chromosome, relocate a gene (unit of alleles) to a different random position within the same chromosome, **shifting** all other genes within the chromosome down one position. This works as follows: first, we randomly determine the gene to relocate, assign it a new random rank_id. Then re-sort the chromosome according to the new ranks with `qsort` with the `qs_partition_rank_id` backend.

Parameters

in	*prob*	prob optional probability of mutation, if absent, the default value commondata::relocation_shift_rate is used.

8.7.3.18.1 Implementation details Do mutate if a random value is smaller than the commondata::relocation_shift_rate parameter constant value.
If yes, randomly determine the gene that initiates move within the chromosome.
Randomly determine the new position of this gene.

Then, cycle through alleles and select new random allele if it happens to coincide with gene_move.
After the cycle, adjust rank_id's of the alleles.
Then re-sort the allele objects by their updated rank_id's.
Finally, recalculate rank_id's so they are again ordered.
Definition at line 772 of file m_genome.f90.

8.7.3.19 genome_init_random()

```
subroutine the_genome::genome_init_random (
             class(individual_genome), intent(inout) this,
             character(len=*), intent(in), optional label )
```

Initialise the genome at random, and set sex as determined by the sex determination locus.

8.7.3.19.1 Implementation details First, create spatial moving object component of the individual genome. But we do not yet position the genome object.
Allocate the genome object, it must have genome_size chromosomes and CHROMOSOME_PLOIDY homologs.
Now cycle over all the chromosomes and homologs and initialise each of them.
On exit from the cycle, set the genome label if provided, or random string if not.
Then, determine the sex of the genome by the genome sex determination locus taking into account the sex ratio.
Definition at line 827 of file m_genome.f90.

8.7.3.20 genome_create_zero()

```
subroutine the_genome::genome_create_zero (
             class(individual_genome), intent(inout) this )
```

Create a new empty genome, and set sex as determined by the sex determination locus. Genome values are from parents using inherit functions.

8.7.3.20.1 Implementation details First of all, Create spatial moving object component of the individual genome.
Allocate the genome object, it must have genome_size chromosomes and CHROMOSOME_PLOIDY homologs
Now cycle over all the chromosomes and homologs and initialise each of them with empty genes (zero).
Initialise the label the genome with a random string.
Determine the sex of the genome by the genome sex determination locus taking into account the sex ratio.
Definition at line 874 of file m_genome.f90.

8.7.3.21 genome_label_set()

```
subroutine the_genome::genome_label_set (
             class(individual_genome), intent(inout) this,
             character(len=*), intent(in), optional label )
```

Label genome. If label is not provided, make a random string.

8.7.3.21.1 Implementation details Set the genome label if provided, or random string if not.
Definition at line 912 of file m_genome.f90.

8.7.3.22 genome_label_get()

```
elemental character(len=label_length) function the_genome::genome_label_get (
             class(individual_genome), intent(in) this )
```

Accessor function to get the genome label. The label is a kind of a (random) text string name of the genome and the individual agent.

Note

We especially need this accessor function because the genome (and individual) name is used in other modules for file names ids etc.

Returns

String label.

Definition at line 935 of file m_genome.f90.

8.7.3.23 genome_sex_determine_init()

```
subroutine the_genome::genome_sex_determine_init (
            class(individual_genome), intent(inout) this )
```

Sex determination initialisation subroutine.
Determine the genome's sex, sex is set by a logical identifier, `sex_is_male` TRUE is **male**. Sex is calculated from the genome and based on the average values of the sex determination alleles in homologous chromosomes, rescaled to 0:1. This rescaled value is then compared with the sex ratio parameter.

8.7.3.23.1 Implementation details The implementation is based on **genotype** x **phenotype** matrix (logical type): commondata::sex_genotype_phenotype.
First, initialise the average sex locus sum across the homologous chromosomes.

8.7.3.23.1.1 Loops Then loop across **homologs**, **chromosomes** and **alleles** until the value of `SEX_GENOTYPE_PHENOTYPE` gets TRUE. This means it is the *sex locus*.

- If this condition is met, set label to the sex locus allele ("SEX_LOCUS").
- Sex is determined by an average of the sex loci of the homologous chromosomes. Therefore, first get the vector of additive allele components.
- And sum it up to finally get the total sum for all chromosomes.
- Finally, also update the counter of the totals.

Upon exit from the loop, check if the average sex locus across all homologous chromosomes and additive allele components, scaled to 0:1 is less than the `SEX_RATIO`, the subject becomes the **male** genotype.
Otherwise, the subject becomes the **female**.
Definition at line 955 of file m_genome.f90.

8.7.3.24 genome_get_sex_is_male()

```
elemental logical function the_genome::genome_get_sex_is_male (
            class(individual_genome), intent(in) this )
```

Get the logical sex ID of the genome object component.

Returns

Returns logical value for sex: is it **male**?

Definition at line 1016 of file m_genome.f90.

8.7.3.25 genome_get_sex_is_female()

```
elemental logical function the_genome::genome_get_sex_is_female (
            class(individual_genome), intent(in) this )
```

Get the logical sex ID of the genome object component.

Returns

Returns logical value for sex: is it a **female**?

Definition at line 1028 of file m_genome.f90.

8.7.3.26 genome_get_sex_label()

```
elemental character(len=label_length) function the_genome::genome_get_sex_label (
            class(individual_genome), intent(in) this )
```

Get the descriptive sex label: male or female.

Returns

Returns the the descriptive sex label

Definition at line 1044 of file m_genome.f90.

8.7.3.27 genome_individual_set_alive()

```
elemental subroutine the_genome::genome_individual_set_alive (
            class(individual_genome), intent(inout) this )
```

Set the individual to be **alive**, normally this function is used after init or birth.
Definition at line 1057 of file m_genome.f90.

8.7.3.28 genome_individual_set_dead()

```
elemental subroutine the_genome::genome_individual_set_dead (
            class(individual_genome), intent(inout) this )
```

Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function.
The `dies` method is implemented at the following levels of the agent object hierarchy (upper overrides the lower level):

- the_genome::individual_genome::dies();
- the_neurobio::appraisal::dies();
- the_neurobio::gos_global::dies();
- the_individual::individual_agent::dies().

Definition at line 1076 of file m_genome.f90.

8.7.3.29 genome_individual_check_alive()

```
elemental logical function the_genome::genome_individual_check_alive (
            class(individual_genome), intent(in) this )
```

Check if the individual is **alive**.
Definition at line 1085 of file m_genome.f90.

8.7.3.30 genome_individual_check_dead()

```
elemental logical function the_genome::genome_individual_check_dead (
            class(individual_genome), intent(in) this )
```

Check if the individual is **dead** (the opposite of `is_alive`).
Definition at line 1095 of file m_genome.f90.

8.7.3.31 genome_individual_recombine_homol_full_rand_alleles()

```
subroutine the_genome::genome_individual_recombine_homol_full_rand_alleles (
            class(individual_genome), intent(inout) this,
            class(individual_genome), intent(in) mother,
            class(individual_genome), intent(in) father,
            real(srp), intent(in), optional exchange_ratio )
```

Internal genetic recombination backend, exchange individual alleles between homologous chromosomes in mother and father genomes to form the `this` (offspring) genome. Fully random recombination.

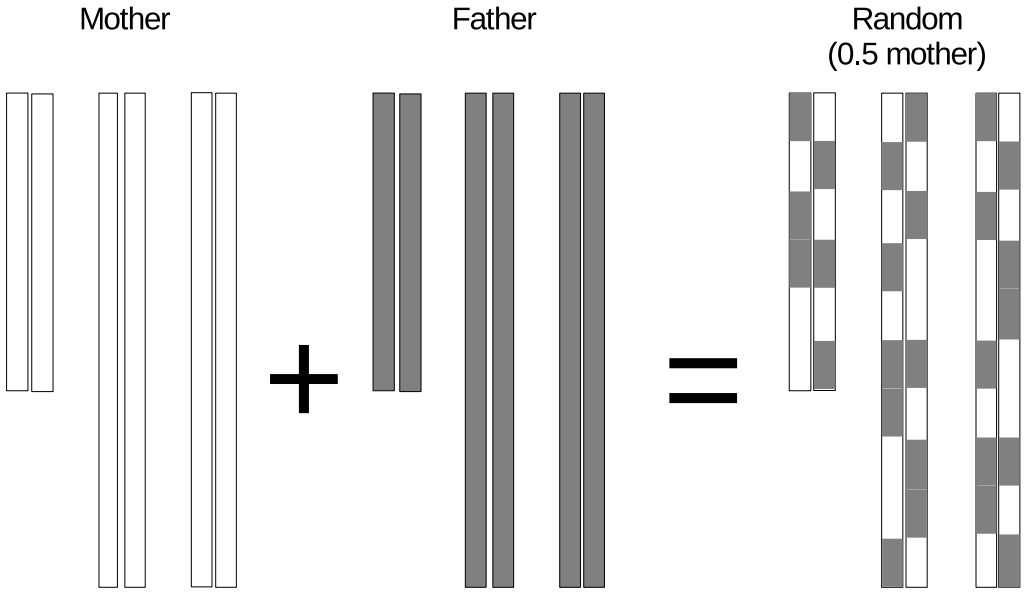


Figure 8.22 Scheme of random genetic recombination

Note

Note that in this procedure, each of the individual alleles is copied from the mother or from the farther **independently** and **randomly**. This means that genetic distances are equal and there is no linkage disequilibrium.

Note also that recombinations across the homologs of the same chromosome are also randomised, i.e. different in all homologs.

Parameters

in	*mother*	mother The **mother** object the_genome::individual_genome class.
in	*father*	father The **father** object the_genome::individual_genome class.

8.7.3.31.1 Notable local variables

- **n_alleles** is the number of alleles for each of the chromosomes, dynamically updated within the loops.

acomp_vect_mother and **acomp_vect_father** are the vectors of additive allele components that are obtained from the mother and the father.

8.7.3.31.2 Implementation details Check optional `exchange_ratio` parameter that defines the ratio of the alleles that are inherited from the mother. If absent, get the default value from the commondata::genome_recombination_ratio_mother parameter.

8.7.3.31.2.1 Nested loops: HOMOLOGS, CHROMOSOMES, ALLELES Loop through the chromosomes (`CHROMOSOMES` block over `i`) and their homologues (`HOMOLOGS` block over `j`) and set the genetic make up of the `this` object from the genes of the `mother` and the `father` objects.

The outline of the main loop is:

- homologues
  - chromosomes
    - alleles

Warning

Note that it is not possible to use parallel `do concurrent` loops here due to non-pure random call of the `RAND_R4()` function in the innermost loop (`ALLELES` block over `k`).

First, get the number of alleles for each of the chromosomes: `n_alleles`.
Then, loop over the `n_alleles` alleles (`ALLELES` block over `k`):
Randomly select if this specific allele (k) is copied from the **mother** or is a subject to (random) recombination and gets the values from the **father**. This is determined stochastically using the `exchange_ratio` ratio dummy argument or the `commondata::genome_recombination_ratio_mother` parameter as the probability to get the allele from the mother.

- Get the vectors of additive allele components for the **mother**.

Finally, **set** the vector of additive allele components of the `this` agent from the mother's vector.

- Get the vectors of additive allele components for the **father**.

Finally, **set** the vector of additive allele components of the `this` agent from the father's vector.
Definition at line 1116 of file m_genome.f90.

8.7.3.32 genome_individual_recombine_homol_part_rand_alleles()

```
subroutine the_genome::genome_individual_recombine_homol_part_rand_alleles (
             class(individual_genome), intent(inout) this,
             class(individual_genome), intent(in) mother,
             class(individual_genome), intent(in) father,
             real(srp), intent(in), optional exchange_ratio )
```

Internal genetic recombination backend, exchange individual alleles between homologous chromosomes in mother and father genomes to form the `this` (offspring) genome. Partially random recombination, identical across the homologous chromosomes.

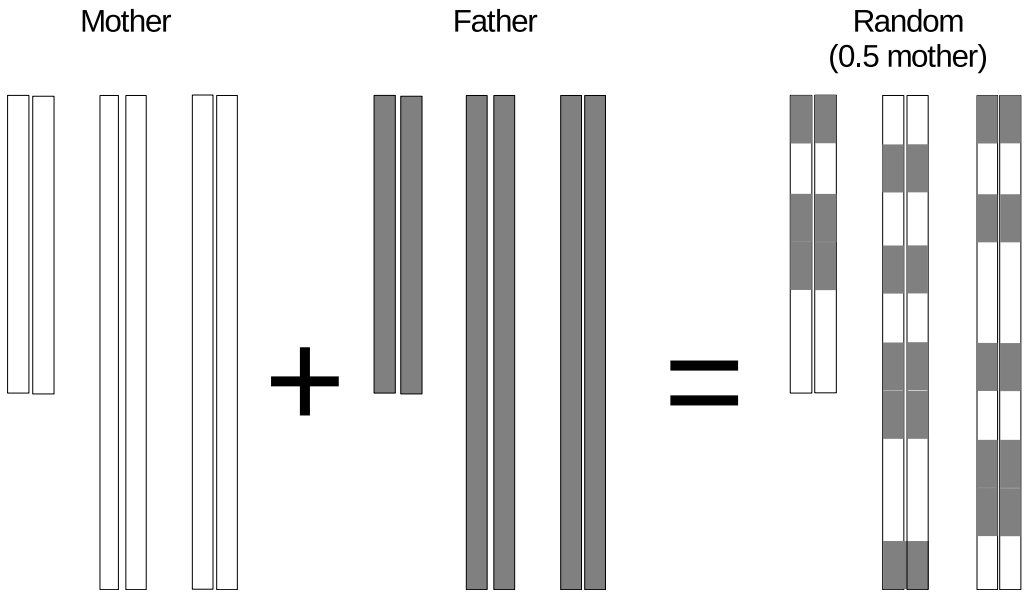


Figure 8.23 Scheme of partially random genetic recombination

Note

Note that in this procedure, each of the individual alleles is copied from the mother or from the farther **independently** and **partially randomly**. This means that genetic distances are equal and there is no linkage disequilibrium.

Note also that recombination is identical across all homologs of the same chromosome.

Parameters

in	*mother*	mother The **mother** object the_genome::individual_genome class.
in	*father*	father The **father** object the_genome::individual_genome class.

8.7.3.32.1 Notable local variables

- **n_alleles** is the number of alleles for each of the chromosomes, dynamically updated within the loops.

acomp_vect_mother and **acomp_vect_father** are the vectors of additive allele components that are obtained from the mother and the father.

8.7.3.32.2 Implementation details Check optional `exchange_ratio` parameter that defines the ratio of the alleles that are inherited from the mother. If absent, get the default value from the commondata::genome_recombination_ratio_mother parameter.

8.7.3.32.2.1 Nested loops: CHROMOSOMES, ALLELES, HOMOLOGS Loop through the chromosomes (`CHROMOSOMES` block over `i`) and their and set the genetic make up of the `this` object from the genes of the `mother` and the `father` objects.
The outline of the main loop is:

- chromosomes
  - alleles
    - homologues (within if block)

Warning

Note that it is not possible to use parallel `do concurrent` loops here due to non-pure random call of the `RAND_R4()` function in the innermost loop (`ALLELES` block over `k`).

First, get the number of alleles for each of the chromosomes: `n_alleles`.
Then, loop over the `n_alleles` alleles (`ALLELES` block over `k`):
Randomly select if this specific allele (k) is copied from the **mother** or is a subject to (random) recombination and gets the values from the **father**. This is determined stochastically using the `exchange_ratio` ratio dummy argument or the `commondata::genome_recombination_ratio_mother` parameter as the probability to get the allele from the mother.
Finally, loop through the homologues of the `i`th chromosome (`HOMOLOGS` block over `j`) and set the same allele across all homologues the same way (transferred from mother or the father). Thus, all homologues have identical non-random recombination pattern, either from the mother or from the father.

- Get the vectors of additive allele components for the **mother**.

Finally, **set** the vector of additive allele components of the `this` agent from the mother's vector.

- Get the vectors of additive allele components for the **father**.

Finally, **set** the vector of additive allele components of the `this` agent from the father's vector.
Definition at line 1227 of file m_genome.f90.

8.7.3.33 genome_individual_crossover_homol_fix()

```
subroutine the_genome::genome_individual_crossover_homol_fix (
            class(individual_genome), intent(inout) this,
            class(individual_genome), intent(in) mother,
            class(individual_genome), intent(in) father,
            logical, dimension(max_nalleles,n_chromosomes), intent(in), optional pattern_↩
matrix )
```

Internal fixed genetic crossover backend, exchange blocks of alleles between homologous chromosomes in mother and father genomes to form the `this` (offspring) genome.

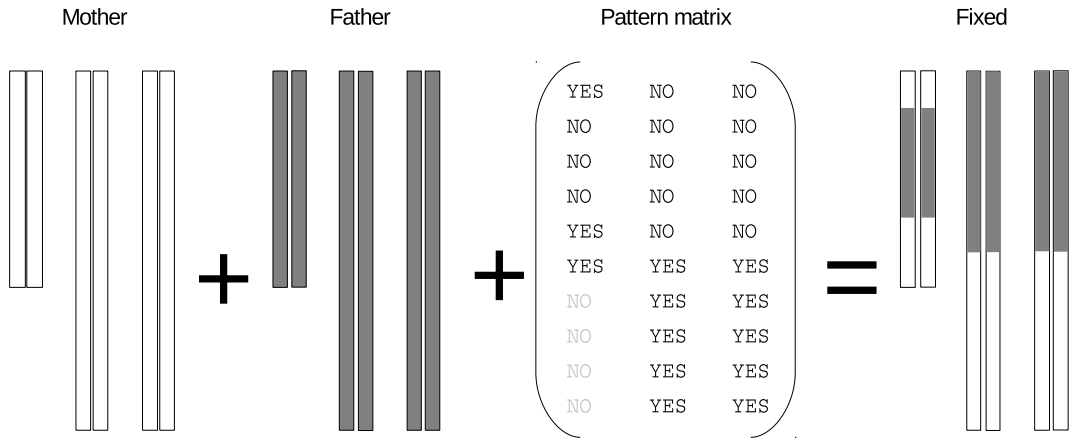


Figure 8.24 Scheme of genetic crossover

Note

Note that in this procedure, blocks of alleles (i.e. sections of the chromosomes) are copied from the mother or from the farther. The blocks are fixed and defined by the boolean parameter matrix commondata::genome_crossover_fixed_mother. This means that genetic recombination is non-random, genetic distances are small within the blocks that are inherited from the same parent and relatively large between the alleles that belong to different blocks (i.e. the genetic distances depend on the proximity of the alleles). This makes linkage disequilibrium possible.

Parameters

in	*mother*	mother The **mother** object the_genome::individual_genome class.
in	*father*	father The **father** object the_genome::individual_genome class.
in	*pattern_matrix*	pattern_matrix is an optional boolean pattern matrix that determines the pattern of fixed chromosome crossover. For each chromosome, the alleles that are marked with the TRUE (YES) values are inherited from the **mother** whereas those marked FALSE (NO) are inherited from the **father**. see commondata::genome_crossover_fixed_mother.

8.7.3.33.1 Notable local variables

- **n_alleles** is the number of alleles for each of the chromosomes, dynamically updated within the loops.

acomp_vect_mother and **acomp_vect_father** are the vectors of additive allele components that are obtained from the mother and the father.

8.7.3.33.2 Implementation details Check optional `pattern_matrix` parameter matrix. If it is not provided, use the default commondata::genome_crossover_fixed_mother.

8.7.3.33.2.1 Nested parallel loops: HOMOLOGS, CHROMOSOMES, ALLELES Loop through the chromosomes (CHROMOSOMES block over i) and their homologues (HOMOLOGSblock over j) and set the genetic make up of the this object from the genes of the mother and the father objects.
First, get the number of alleles for each of the chromosomes: n_alleles.
Then, loop over the n_alleles alleles (ALLELES block over k):
Check the boolean pattern matrix pattern_matrix_locvalue, if it is TRUE, the allele value is copied from the **mother**, otherwise from the **father**.

If block check

- Get the vectors of additive allele components for the **mother**.

Finally, **set** the vector of additive allele components of the this agent from the mother's vector.

- Get the vectors of additive allele components for the **father**.

Finally, **set** the vector of additive allele components of the this agent from the father's vector.
Definition at line 1347 of file m_genome.f90.

8.7.3.34 trait_init_genotype_gamma2gene()

```
subroutine the_genome::trait_init_genotype_gamma2gene (
             class(individual_genome), intent(inout) this,
             real(srp), intent(out) this_trait,
             logical, dimension(:,:), intent(in) g_p_matrix,
             real(srp), intent(in) init_val,
             real(srp), intent(in), optional gerror_cv,
             character(len=*), intent(in) label )
```

Initialise an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy.
Create a new **trait** object from the genotype according to the boolean genotype x phenotype matrix g_p_matrix by commondata::gamma2gene() function and the init_val baseline value. It also introduces an initial Gaussian variance with the coefficient of variation=gerror_cv. Normally, g_p_matrix, init_val and gerror_cv are parameters set in commondata for each specific trait. For example, for *thyroid* hormone (the_hormones::hormones class level up) they are:

- commondata::thyroid_genotype_phenotype,
- commondata::thyroid_init and
- commondata::thyroid_gerror_cv.

Note

This procedure has the **intent [inout]** for this, the agent as it modifies the label. Therefore, it should be mainly used for initialisation of agent traits.

Warning

This code does not (yet) work with *haploid* genotype commondata::chromosome_ploidy = 1, in such a case there is no need to select random homologous chromosome and it is impossible to set the two parameters of the commondata::gamma2gene function (there is only a single chromosome).

Parameters

out	*this_trait*	this_trait the component of the individual agent object hierarchy that we are initialising in.
in	*g_p_matrix*	g_p_matrix matrix genotype x phenotype, the matrix setting the correspondence between the genome and the phenotype.

Parameters

in	*init_val*	init_val start value, trait initialisation baseline value parameter, e.g. for *thyroid* it is commondata::thyroid_init.
in	*gerror_cv*	gerror_cv error value, trait Gaussian error value, e.g. for *thyroid* it is commondata::thyroid_gerror_cv, if absent, we don't introduce random error and the initial trait values are deterministic by the genome.
in	*label*	label a label for the allele locus that sets the phenotypic object.

8.7.3.34.1 Implementation details Set **trait** values from the genome using the g_p_matrix matrix. We first need to select two chromosomes from the available set (normally two, but possibly more) for input into commondata::gamma2gene() function. We do it random.
Then, cycle through and select a different chromosomes (i.e. cycle if happens to coincide with the first). (Use do while chromosome1 = chromosome2 construct.)
As a result there are two distinct chromosomes k1 and k2. This unique chromosomes selection part of the code precludes the use of haploid genome.

8.7.3.34.1.1 Loop over chromosomes and alleles

- set label to the **trait** locus allele
- The initial trait value is determined using the commondata::gamma2gene() function using additive allele components. So, we first get the two vectors of additive allele components.
- And get the value of the trait from commondata::gamma2gene().

Note

commondata::gamma2gene() now accepts vectors of additive allele components.

Definition at line 1471 of file m_genome.f90.

8.7.3.35 trait_set_genotype_gamma2gene()

```
subroutine the_genome::trait_set_genotype_gamma2gene (
             class(individual_genome), intent(in) this,
             real(srp), intent(out) this_trait,
             logical, dimension(:,:), intent(in) g_p_matrix,
             real(srp), intent(in) init_val,
             real(srp), intent(in), optional gerror_cv )
```

Set an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy.
This is almost the same as the_genome::trait_init_genotype_gamma2gene(), but does *not* modify the this object (it has **intent [in]**). Therefore it should be used for setting such traits as behavioural expectancies.

Note

This procedure has the intent [in] for this, the agent as it does not modify the object (e.g. does not set label). Therefore, it can be used in assessing the subjective expectancies.

Warning

This code does not (yet) work with *haploid* genotype commondata::chromosome_ploidy = 1, in such a case there is no need to select random homologous chromosome and it is impossible to set the two parameters of the commondata::gamma2gene() function (there is only a single chromosome).

Parameters

out	*this_trait*	this_trait the component of the individual agent object hierarchy that we are initialising.
in	*g_p_matrix*	matrix genotype x phenotype, the matrix setting the correspondence between the genome and the phenotype.
in	*init_val*	init_val start value, trait initialisation baseline value parameter, e.g. for *thyroid* it is `commondata::thyroid_init`.
in	*gerror_cv*	gerror_cv error value, trait Gaussian error value, e.g. for *thyroid* it is `commondata::thyroid_gerror_cv`, if absent, we don't introduce random error and the initial trait values are deterministic by the genome.

8.7.3.35.1 Implementation details Set **trait** values from the genome using the `g_p_matrix` matrix. We first need to select two chromosomes from the available set (normally two, but possibly more) for input into commondata::gamma2gene() function. We do it random.
Then, we cycle through and select a different chromosomes (i.e. cycle if happens to coincide with the first.
As a result there are two distinct chromosomes `k1` and `k2`. This unique chromosomes selection part of the code precludes the use of haploid genome.

8.7.3.35.1.1 Loop over chromosomes and alleles

- The initial trait value is determined using the commondata::gamma2gene() function using additive allele components. So, we first get the two vectors of additive allele components.
- And get the value of the trait from `commondata::gamma2gene()`.

Note

commondata::gamma2gene() now accepts vectors of additive allele components.

Definition at line 1561 of file m_genome.f90.

8.7.3.36 trait_init_linear_sum_additive_comps_2genes_r()

```
subroutine the_genome::trait_init_linear_sum_additive_comps_2genes_r (
             class(individual_genome), intent(inout) this,
             real(srp), intent(out) this_trait,
             logical, dimension(:,:), intent(in) g_p_matrix,
             real(srp), intent(in) phenotype_min,
             real(srp), intent(in) phenotype_max,
             character(len=*), intent(in) label )
```

Initialise an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy.
Create a new **trait** object from the genotype according to the boolean genotype x phenotype matrix `g_p_matrix` by a simple linear scaling transformation function. Normally, `g_p_matrix`, is a parameters set in commondata for each specific trait.

Note

This version uses only two chromosomes for compatibility with the counterpart the_genome::trait_init_genotype_gamma2gene() procedure; the code logic is almost the same as in that based on the neuronal response function.

Parameters

out	*this_trait*	this_trait the component of the individual agent object hierarchy that we are initialising in.
in	*g_p_matrix*	g_p_matrix matrix genotype x phenotype, the matrix setting the correspondence between the genome and the phenotype.

Parameters

in	*phenotype_min*	init_val start value, trait initialisation baseline value parameter, e.g. for *thyroid* it is commondata::thyroid_init.
in	*label*	label a label for the allele locus that sets the phenotypic object.

8.7.3.36.1 Implementation details Set **trait** values from the genome using the g_p_matrix matrix. We first need to select two chromosomes from the available set (normally two, but possibly more) for input into gamma2gene. We do it random. Assign the first chromosome.
Also assign the second chromosome.
Then, cycle through and select a different chromosomes (i.e. cycle if happens to coincide with the first). (Use do while chromosome1 = chromosome2` construct.)
As a result there are two distinct chromosomes k1 and k2. This unique chromosomes selection part of the code precludes the use of haploid genome.

8.7.3.36.1.1 Loop over chromosomes and alleles

- set label to the **trait** locus allele
- The initial trait value is determined using the gamma2gene function using additive allele components. So, we first get the two vectors of additive allele components.
- Unlike trait_init_genotype_gamma2gene, the output phenotypic value is obtained by simple linear commondata::rescale from the minimum genotype range to the phenotypic range set by the phenotype↩ _min and phenotype_max parameters.

Definition at line 1646 of file m_genome.f90.

8.7.3.37 trait_set_linear_sum_additive_comps_2genes_r()

```
subroutine the_genome::trait_set_linear_sum_additive_comps_2genes_r (
             class(individual_genome), intent(in) this,
             real(srp), intent(out) this_trait,
             logical, dimension(:,:), intent(in) g_p_matrix,
             real(srp), intent(in) phenotype_min,
             real(srp), intent(in) phenotype_max )
```

Set an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy.
This is almost the same as the_genome::trait_init_linear_sum_additive_comps_2genes_r(), but does *not* modify the this object (it has **intent [in]**). Therefore it should be used for setting such traits as behavioural expectancies. Create a new **trait** object from the genotype according to the boolean genotype x phenotype matrix g_p_matrix by a simple linear scaling transformation function. Normally, g_p_matrix, is a parameters set in commondata for each specific trait.

Note

This version uses only two chromosomes for compatibility with the counterpart the_genome::trait_init_genotype_gamma2gene() procedure; the code logic is almost the same as in that based on the neuronal response function.

Parameters

out	*this_trait*	this_trait the component of the individual agent object hierarchy that we are initialising in.
in	*g_p_matrix*	g_p_matrix matrix genotype x phenotype, the matrix setting the correspondence between the genome and the phenotype.
in	*phenotype_min*	init_val start value, trait initialisation baseline value parameter, e.g. for *thyroid* it is commondata::thyroid_init.

8.7.3.37.1 Implementation details Set **trait** values from the genome using the `g_p_matrix` matrix. We first need to select two chromosomes from the available set (normally two, but possibly more) for input into `gamma2gene`. We do it random. Assign the first chromosome.
Also assign the second chromosome.
Then, cycle through and select a different chromosomes (i.e. cycle if happens to coincide with the first). (Use do while chromosome1 = chromosome2` construct.)
As a result there are two distinct chromosomes `k1` and `k2`. This unique chromosomes selection part of the code precludes the use of haploid genome.

8.7.3.37.1.1 Loop over chromosomes and alleles

- The initial trait value is determined using the gamma2gene function using additive allele components. So, we first get the two vectors of additive allele components.
- Unlike trait_init_genotype_gamma2gene, the output phenotypic value is obtained by simple linear commondata::rescale from the minimum genotype range to the phenotypic range set by the `phenotype_min` and `phenotype_max` parameters.

Definition at line 1742 of file m_genome.f90.

8.7.3.38 genome_mutate_wrapper()

```
subroutine the_genome::genome_mutate_wrapper (
             class(individual_genome), intent(inout) this,
             real(srp), intent(in), optional p_point,
             real(srp), intent(in), optional p_set,
             real(srp), intent(in), optional p_swap,
             real(srp), intent(in), optional p_shift )
```

Perform a probabilistic random mutation(s) on the individual genome. This is a high level wrapper to build mutations from various components.

Parameters

in	*p_point*	p_point optional probability of mutation, if absent, the default value commondata::mutationrate_point is used.
in	*p_set*	p_set optional probability of mutation, if absent, the default value commondata::mutationrate_batch is used.
in	*p_swap*	p_swap optional probability of mutation, if absent, the default value commondata::mutationrate_batch is used.
in	*p_shift*	p_shift optional probability of mutation, if absent, the default value commondata::mutationrate_batch is used.

8.7.3.38.1 Implementation notes Each of the mutations below is a random process that occurs with a specific probability that is set by its respective mutation rate parameter.

- Call for a point mutation on a single random allele component of a randomly chosen chromosome using the the_genome::gene::mutate_point() method.

Call for a point mutation on a whole random allele (batch of allele components) of a randomly chosen chromosome using the the_genome::gene::mutate_set() method.
Definition at line 1820 of file m_genome.f90.

8.7.4 Variable Documentation

8.7.4.1 modname

```
character (len=*), parameter, private the_genome::modname = "(THE_GENOME)"  [private]
```
Definition at line 27 of file m_genome.f90.

8.8 the_hormones Module Reference

Definition the hormonal architecture of the agent.

Data Types

- type hormones

  This type adds hormonal architecture extending the genome object.

Functions/Subroutines

- subroutine hormones_init_genotype (this)

  Initialise hormone levels based on the genome value. Two alleles are selected at random and input into the `gamma2gene` *function to get the initial hormone values rescaled to 0:1. Note that the* `gamma2gene` *alleles defining the* ***shape*** *of the gamma function and the* ***half-max effect*** *are selected randomly in this version. Also, polyploid organisms are possible, in such case, two parameters are also randomly defined from a larger set (e.g. from four chromosomes in case of tetraploids). See implementation details and comments for each of the hormones.*
- elemental subroutine hormones_clean_history_stack (this)

  Clean the history stack of hormones: testosterone and estrogen histories are set to `MISSING`*.*
- elemental subroutine hormones_update_history (this)

  Update the sex steroid hormones history stack from the current level.
- elemental real(srp) function hormones_reproductive_factor_calc (this)

  Calculate the reproductive factor. Reproductive factor is defined as the current level of the_hormones::testosterone_level in males and the_hormones::estrogen_level in females.
- elemental real(srp) function testosteron_baseline_get_level (this)

  Get the value of testosterone baseline.
- elemental real(srp) function estrogen_baseline_get_level (this)

  Get the value of estrogen baseline.

Accessor functions for all the hormones.

Get and set functions for each hormone follow. We left them as individual hormone-specific functions duplicating code. Not ideal, but easy to use provided hormones do not change too often. Tiny atomic hormone get/set functions are easy to code.

- elemental real(srp) function growhorm_get_level (this)

  Get the value of ***growth hormone****.*
- elemental subroutine growhorm_set_level (this, value_set)

  Set the value of ***growth hormone****.*
- elemental real(srp) function thyroid_get_level (this)

  Get the value of ***thyroid****.*
- elemental subroutine thyroid_set_level (this, value_set)

  Set the value of ***thyroid****.*
- elemental real(srp) function adrenaline_get_level (this)

  Get the value of ***adrenaline****.*
- elemental subroutine adrenaline_set_level (this, value_set)

  Set the value of ***adrenaline****.*
- elemental real(srp) function cortisol_get_level (this)

  Get the value of ***cortisol****.*
- elemental subroutine cortisol_set_level (this, value_set)

  Set the value of ***cortisol****.*
- elemental real(srp) function testosterone_get_level (this)

  Get the value of ***testosterone****.*

- elemental subroutine testosterone_set_level (this, value_set, update_history)
  *Set the value of **testosterone**.*
- elemental real(srp) function estrogen_get_level (this)
  *Get the value of **estrogen**.*
- elemental subroutine estrogen_set_level (this, value_set, update_history)
  *Set the value of **estrogen**.*

Variables

- character(len= ∗), parameter, private modname = "(THE_HORMONES)"

8.8.1 Detailed Description

Definition the hormonal architecture of the agent.

8.8.2 THE_HORMONES module

Define the hormonal architecture objects. Hormones affect implemented in such a way as to affect decision making and behaviour, but change relatively slowly across the lifespan of the agent. Their initial state is also genetically determined.

8.8.3 Function/Subroutine Documentation

8.8.3.1 hormones_init_genotype()

```
subroutine the_hormones::hormones_init_genotype (
            class(hormones), intent(inout) this )
```

Initialise hormone levels based on the genome value. Two alleles are selected at random and input into the `gamma2gene` function to get the initial hormone values rescaled to 0:1. Note that the `gamma2gene` alleles defining the **shape** of the gamma function and the **half-max effect** are selected randomly in this version. Also, polyploid organisms are possible, in such case, two parameters are also randomly defined from a larger set (e.g. from four chromosomes in case of tetraploids). See implementation details and comments for each of the hormones.

8.8.3.1.1 Implementation details First, get all the initial hormone level values from the genotype.
Then, initialise the baseline levels of sex steroids from the starting genetically determined hormone levels.
Clean history stack of all the hormones upon init.
Finally, update the hormone history stack with the first init values.
Definition at line 148 of file m_hormon.f90.

8.8.3.2 hormones_clean_history_stack()

```
elemental subroutine the_hormones::hormones_clean_history_stack (
            class(hormones), intent(inout) this )
```

Clean the history stack of hormones: testosterone and estrogen histories are set to `MISSING`.
Definition at line 195 of file m_hormon.f90.

8.8.3.3 hormones_update_history()

```
elemental subroutine the_hormones::hormones_update_history (
            class(hormones), intent(inout) this )
```

Update the sex steroid hormones history stack from the current level.
Update the hormone history stack with the first init values.
Definition at line 205 of file m_hormon.f90.

8.8.3.4 hormones_reproductive_factor_calc()

```
elemental real(srp) function the_hormones::hormones_reproductive_factor_calc (
            class(hormones), intent(in) this )
```

Calculate the reproductive factor. Reproductive factor is defined as the current level of the_hormones↩ ::testosterone_level in males and the_hormones::estrogen_level in females.

Note

Because the reproductive factor is obtained by sex-specific operations directly on the hormones, the use of this function is mostly limited to diagnostic outputs.

Returns

Reproductive factor.

Definition at line 221 of file m_hormon.f90.

8.8.3.5 growhorm_get_level()

```
elemental real(srp) function the_hormones::growhorm_get_level (
            class(hormones), intent(in) this )
```

Get the value of **growth hormone**.

Returns

value, Returns the value of the **growth hormone**.

Definition at line 245 of file m_hormon.f90.

8.8.3.6 growhorm_set_level()

```
elemental subroutine the_hormones::growhorm_set_level (
            class(hormones), intent(inout) this,
            real(srp), intent(in) value_set )
```

Set the value of **growth hormone**.

Parameters

in	*value_set*	value, Set the value of the **growth hormone**.

Definition at line 257 of file m_hormon.f90.

8.8.3.7 thyroid_get_level()

```
elemental real(srp) function the_hormones::thyroid_get_level (
            class(hormones), intent(in) this )
```

Get the value of **thyroid**.

Returns

value, Returns the value of the **thyroid hormone**.

Definition at line 269 of file m_hormon.f90.

8.8.3.8 thyroid_set_level()

```
elemental subroutine the_hormones::thyroid_set_level (
```

```
class(hormones), intent(inout) this,
real(srp), intent(in) value_set )
```

Set the value of **thyroid**.

Parameters

in	*value_set*	value, Set the value of the **thyroid hormone**.

Definition at line 281 of file m_hormon.f90.

8.8.3.9 adrenaline_get_level()

```
elemental real(srp) function the_hormones::adrenaline_get_level (
class(hormones), intent(in) this )
```

Get the value of **adrenaline**.

Returns

value, Returns the value of **adrenaline**.

Definition at line 293 of file m_hormon.f90.

8.8.3.10 adrenaline_set_level()

```
elemental subroutine the_hormones::adrenaline_set_level (
class(hormones), intent(inout) this,
real(srp), intent(in) value_set )
```

Set the value of **adrenaline**.

Parameters

in	*value_set*	value, Set the value of the **adrenaline**.

Definition at line 305 of file m_hormon.f90.

8.8.3.11 cortisol_get_level()

```
elemental real(srp) function the_hormones::cortisol_get_level (
class(hormones), intent(in) this )
```

Get the value of **cortisol**.

Returns

value, Returns the value of **cortisol**

Definition at line 317 of file m_hormon.f90.

8.8.3.12 cortisol_set_level()

```
elemental subroutine the_hormones::cortisol_set_level (
class(hormones), intent(inout) this,
real(srp), intent(in) value_set )
```

Set the value of **cortisol**.

Parameters

in	*value_set*	value, Set the value of the **cortisol**.

Definition at line 329 of file m_hormon.f90.

8.8.3.13 testosterone_get_level()

```
elemental real(srp) function the_hormones::testosterone_get_level (
            class(hormones), intent(in) this )
```

Get the value of **testosterone**.

Returns

value, Returns the value of **testosterone**

Definition at line 341 of file m_hormon.f90.

8.8.3.14 testosterone_set_level()

```
elemental subroutine the_hormones::testosterone_set_level (
            class(hormones), intent(inout) this,
            real(srp), intent(in) value_set,
            logical, intent(in), optional update_history )
```

Set the value of **testosterone**.

Parameters

in	*value_set*	value_set, Set the value of the **testosterone**.
in	*update_history*	update_history is an optional logical flag to update the hormone history stack, the default is to do update, no update only if explicitly set to FALSE.

Definition at line 353 of file m_hormon.f90.

8.8.3.15 estrogen_get_level()

```
elemental real(srp) function the_hormones::estrogen_get_level (
            class(hormones), intent(in) this )
```

Get the value of **estrogen**.

Returns

value, Returns the value of **estrogen**

Definition at line 375 of file m_hormon.f90.

8.8.3.16 estrogen_set_level()

```
elemental subroutine the_hormones::estrogen_set_level (
            class(hormones), intent(inout) this,
            real(srp), intent(in) value_set,
            logical, intent(in), optional update_history )
```

Set the value of **estrogen**.

Parameters

in	*value_set*	value, Set the value of the **estrogen**.
in	*update_history*	update_history is an optional logical flag to update the hormone history stack, the default is to do update, no update only if explicitly set to FALSE.

Definition at line 387 of file m_hormon.f90.

8.8.3.17 testosteron_baseline_get_level()

```
elemental real(srp) function the_hormones::testosteron_baseline_get_level (
            class(hormones), intent(in) this )
```

Get the value of testosterone baseline.

Returns

value, Returns the value of **testosterone** baseline.

Definition at line 410 of file m_hormon.f90.

8.8.3.18 estrogen_baseline_get_level()

```
elemental real(srp) function the_hormones::estrogen_baseline_get_level (
            class(hormones), intent(in) this )
```

Get the value of estrogen baseline.

Returns

value, Returns the value of **testosterone** baseline.

Definition at line 422 of file m_hormon.f90.

8.8.4 Variable Documentation

8.8.4.1 modname

```
character (len=*), parameter, private the_hormones::modname = "(THE_HORMONES)"  [private]
```

Definition at line 25 of file m_hormon.f90.

8.9 the_individual Module Reference

An umbrella module that collects all the components of the individual agent.

Data Types

- type individual_agent

  This type describes parameters of the individual agent.

Functions/Subroutines

- subroutine, private individual_init_random (this, exclude_genome)

  Generate a random agent from the genotype.
- subroutine individual_create_zero (this)

  Generate a new empty agent.
- elemental subroutine individual_agent_set_dead (this)

  *Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function.*
- subroutine kill_destroy (this)

  Finalisation procedure. Note that finalisation of objects may not yet be implemented in the compiler. Therefore this subroutine is not used so far, just a stub.
- elemental real(srp) function individual_get_risk_mortality_individual (this)

Get the individually-specific mortality risk for the agent.

- elemental subroutine individual_preevol_fitness_calc (this)

  Calculate fitness for the pre-evolution phase of the genetic algorithm. Pre-evolution is based on selection for a simple criterion without explicit reproduction etc. The criterion for selection at this phase is set by the integer the_individual::individual_agent::fitness component.

Variables

- character(len= ∗), parameter, private modname = "(THE_INDIVIDUAL)"

8.9.1 Detailed Description

An umbrella module that collects all the components of the individual agent.

8.9.2 THE_INDIVIDUAL module

Define the individual fish object and its properties and methods/functions

8.9.3 Function/Subroutine Documentation

8.9.3.1 individual_init_random()

```
subroutine, private the_individual::individual_init_random (
             class(individual_agent), intent(inout) this,
             logical, intent(in), optional exclude_genome )  [private]
```

Generate a random agent from the genotype.
This subroutine is used to initialise the individual agent with random data from the genotype. Its use is the most natural for the initialisation of a population of agents. The init procedure calls the init functions for the lower order layers of the class hierarchy (genome, hormones, neurobio), and sets values.

Parameters

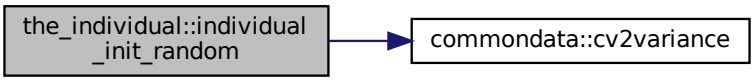
in	*exclude_genome*	exclude_genome is a logical flag to exclude initialisation of random genome. If absent, assumed FALSE.

Note

Only the first generation is initialised with random genome (exclude_genome = FALSE), all subsequent generations use pre-existing genomes from the ancestors (exclude_genome = TRUE) randomised by crossover in the genetic algorithm.

We first initialise all the components of the agent down the class hierarchy: from individual genome to the neurobiological architecture.
Clean the spatial location history stack of the agent.
initialise hormone objects **from genome**
initialise condition **from genome**
initialise empty reproduction objects
initialise empty neuro objects
Finally, we bring the agent to life by setting alive boolean flag
Set the individually specific mortality risk initially as a Gaussian variable with mean commondata::individual_mortality_risk_def and CV commondata::individual_mortality_risk_cv. There is also a restriction that the risk of mortality should never be smaller than commondata::zero.
Calculate the initial value of fitness for pre-evolution stage.
Definition at line 94 of file m_indiv.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

8.9.3.2 individual_create_zero()

```
subroutine the_individual::individual_create_zero (
            class(individual_agent), intent(inout) this )
```

Generate a new empty agent.
This subroutine is used to create a new empty individual agent. It should be used to make newborn agents that inherit traits from their parents.

Warning

This procedure is not used so far and is candidate for being deprecated. Not clear if necessary.

Clean the spatial location history stack of the agent.
create **empty** genome
Create empty hormone objects. No genome-based initialisation is done.
Empty condition objects. Clean history stack.
Initialise reproduction objects and neurobiology to a zero state.
Finally, we bring the agent to life by setting alive boolean flag
Calculate the initial value of fitness for pre-evolution stage.
Definition at line 148 of file m_indiv.f90.

8.9.3.3 individual_agent_set_dead()

```
elemental subroutine the_individual::individual_agent_set_dead (
            class(individual_agent), intent(inout) this )
```

Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function.
The `dies` method is implemented at the following levels of the agent object hierarchy (upper overrides the lower level):

- the_genome::individual_genome::dies();
- the_neurobio::appraisal::dies();

- the_neurobio::gos_global::dies();
- the_individual::individual_agent::dies().

Note

This method overrides the the_genome::individual_genome::dies() method, nullifying all reproductive and neurobiological and behavioural objects.

- Set the agent "dead";
- emptify reproduction objects;
- emptify neurobiological objects.

Definition at line 203 of file m_indiv.f90.

8.9.3.4 kill_destroy()

```
subroutine the_individual::kill_destroy (
            class(individual_agent), intent(inout) this )
```

Finalisation procedure. Note that finalisation of objects may not yet be implemented in the compiler. Therefore this subroutine is not used so far, just a stub.
Definition at line 216 of file m_indiv.f90.

8.9.3.5 individual_get_risk_mortality_individual()

```
elemental real(srp) function the_individual::individual_get_risk_mortality_individual (
            class(individual_agent), intent(in) this )
```

Get the individually-specific mortality risk for the agent.

Returns

Individually-specific mortality risk for the agent.

Definition at line 228 of file m_indiv.f90.

8.9.3.6 individual_preevol_fitness_calc()

```
elemental subroutine the_individual::individual_preevol_fitness_calc (
            class(individual_agent), intent(inout) this )
```

Calculate fitness for the pre-evolution phase of the genetic algorithm. Pre-evolution is based on selection for a simple criterion without explicit reproduction etc. The criterion for selection at this phase is set by the integer the_individual::individual_agent::fitness component.

Warning

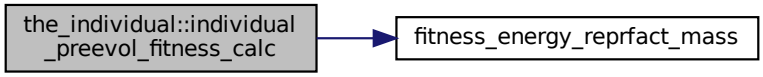
Note that fitness here is actually an "antifitness", the higher its value, the **worse** fitting is the agent.

INT_FITNESS_DEAD is the fitness ascribed to the dead agent, it should be a very large value greater than for any alive.
First check if the agent is dead. If so, give very high value that is never selected.
Now, the reverse of fitness can be calculated by various methods. Specific calculation functions are implemented within this function. So far the following routines were implemented:

- fitness_energy_reprfact_mass
- fitness_energy_reprfact())
- fitness_mass_incr_abs()

- fitness_birth_mass_ratio()
- fitness_stomach_mass_ratio()
- fitness_stomach_mass_abs()
- fitness_food_mass_sum()
- fitness_mass_incr_ratio()
- fitness_reprod_factor()

Definition at line 245 of file m_indiv.f90.
Here is the call graph for this function:



8.9.4 Variable Documentation

8.9.4.1 modname

```
character (len=*), parameter, private the_individual::modname = "(THE_INDIVIDUAL)"  [private]
```

Definition at line 25 of file m_indiv.f90.

8.10 the_neurobio Module Reference

Definition of the decision making and behavioural the architecture.

Data Types

- type percept_food

  This type defines how the agent perceives food items. The food perception object the_neurobio::percept_food is basically an array of food objects within the visual range of the agent plus distances to the agent. This is the "objective" perception container, reflecting the "real world". We introduce a perception error when perception object is analysed by the agent's neurobiological system.
- type spatial_percept_component

  This type defines a single spatial perception component, i.e. some single elementary spatial object that can be perceived by the agent from a big array of objects of the same type which are available in the agent's environment. Different kinds of perception objects (e.g. conspecifics, predators etc.) can be produced by extending this basic type.
- type conspec_percept_comp

  *This type defines a **single conspecific** perception component. It is required for the the_neurobio::percept_conspecifics type that defines the whole conspecifics perception object (array of conspecifics).*
- type percept_conspecifics

  This type defines how the agent perceives conspecifics.
- type spatialobj_percept_comp

  *This type defines a **single** arbitrary spatial object perception component. For example, a predator perception object is then an array of such spatial object perception components.*
- type percept_predator

  This type defines how the agent perceives a predator.

- type percept_stomach

  This type defines how the agent perceives its own stomach capacity.
- type percept_body_mass

  This type defines how the agent perceives its own body mass it can be important for state-dependency.
- type percept_energy

  This type defines how the agent perceives its own energy reserves it can be important for state-dependency.
- type percept_age

  This type defines how the agent perceives its own age in terms of the model discrete time step.
- type percept_reprfact

  Perception of the reproductive factor, reproductive factor depends on the sex hormones differently in males and females.
- type percept_light

  Perception of the ambient illumination. This is a very simple perception component, singular and static.
- type percept_depth

  Perception of the current depth horizon.
- type memory_perceptual

  Individual perception memory(history) stack, a memory component that saves perception values at previous time steps of the model. Not whole perception objects are saved for simplicity, only the most important parameters, integer and real types so commondata::add_to_history() can be used in unmodified form. Decision making can make use of this memory stack.
- type perception

  *The perception architecture of the agent. See "The perception mechanism" for a general overview. At this level, lower order perception objects are combined into the the_neurobio::perception class hierarchy level of the agent. The object bound functions see_ and feel_ obtain (**set**) the specific perception objects from the external or internal environments of the agent and put them into the the_neurobio::perception data structure. Also, memory component is updated with the perception data. Perception objects can then be used as input into the individual decision-making procedures.*
- type percept_components_motiv

  Perceptual components of motivational states. Plugged into all `STATE_`, attention etc. These components are linked to specific inner or outer perception objects (stimuli). Their sum result(s) in the overall value of the motivation component.
- type state_motivation_base

  *These types describe the **neurobiological states** of the agent. (1) Each state may have several components that are related to specific inner or outer perception objects (stimuli). (2) There is also a `motivation` component that describes the global **motivation** value for this state.*
- interface motivation_init_root

  *Abstract interface for the deferred **init** function `clean_init` that has to be overridden by each object that extends the basic motivational state type.*
- type state_hunger

  *The motivational state of **hunger**. Evokes food seeking, eating, higher activity, emigrating and habitat switching.*
- type state_fear_defence

  *The state of **fear state**. Evokes active escape, fleeing, emigration and habitat switch.*
- type state_reproduce

  *The state of **reproduction**. Evokes seeking conspecifics and mating during the reproductive phase.*
- type motivation

  ***Motivation** is a collection of all internal motivational states of the agent. This type is also used in defining Expectancies of motivations.*
- type memory_emotional

  *Individual motivation/emotion memory stack, a memory component that saves the values of the **final motivations** at previous time steps of the model. Not whole state (`STATE_`) objects are saved for simplicity. `add_to_history` is used in unmodified form. Decision making can make use of this emotional memory stack.*
- type appraisal

  *The **appraisal** level. At this level, perception objects are feed into the commondata::gamma2gene() sigmoid function and the neuronal responses are obtained at the output. Neuronal responses for different perception objects are then summed up and the promary motivation values are obtained. Following this, modulation alters some of the primary motivation values resulting in the final motivation values. See "From perception to GOS" for an overview.*

- type gos_global

  Global organismic state (GOS) level. GOS is defined by the dominant motivational state component (`STATE_`), namely, by the logical flag %dominant_state. If this logical flag is TRUE for a particular motivational state component, this state is the GOS. Thus, there should be is no separate data component(s) e.g. "value" for GOS. The values the_neurobio::gos_global::gos_main and the_neurobio::gos_global::gos_arousal can be inferred from the motivations, here are doubled mainly for convenience. See "From perception to GOS" for an overview.

Functions/Subroutines

- elemental subroutine percept_food_create_init (this, maximum_number_food_items)

  *Initiate an empty **food** perception object with known number of components.*
- subroutine percept_food_number_seen (this, number_set)

  Set the total number of food items perceived (seen) in the food perception object. Do not reallocate the perception object components with respect to this new number yet.
- subroutine percept_food_make_fill_arrays (this, items, dist)

  Make the food perception object, fill it with the actual data arrays.
- elemental integer function percept_food_get_count_found (this)

  Get the number (count) of food items seen. Trivial.
- elemental real(srp) function percept_food_get_meansize_found (this)

  Get the average size of food items seen. Trivial.
- elemental real(srp) function percept_food_get_meanmass_found (this)

  Get the average mass of food items seen. Trivial.
- elemental real(srp) function percept_food_get_meandist_found (this)

  Get the average distance to the food items seen. Trivial.
- elemental subroutine percept_food_destroy_deallocate (this)

  *Deallocate and delete a **food** perception object.*
- subroutine food_perception_get_visrange_objects (this, food_resource_available, time_step_model)

  Get available food items within the visual range of the agent, which the agent can perceive and therefore respond to. Food perception is packaged into the food perception object this%perceive_food for output.
- elemental logical function food_perception_is_seeing_food (this)

  Check if the agent sees any food items within its visual range.
- real(srp) function food_perception_probability_capture_memory_object (this, last, time_step_model)

  Calculate the probability of capture of a subjective representation of food item based on the data from the perceptual memory stack.
- elemental subroutine percept_stomach_create_init (this)

  *Initiate an empty **stomach** capacity perception object.*
- elemental real(srp) function percept_stomach_get_avail_capacity (this)

  *Get the currently available value of the available **stomach** volume.*
- subroutine percept_stomach_update_avail_capacity (this, current_volume)

  *Set and update the currently available value of the available **stomach** volume.*
- elemental subroutine percept_stomach_destroy_deallocate (this)

  *Destroy the **stomach** perception object and deallocate it.*
- elemental subroutine percept_bodymass_create_init (this)

  *Initiate an empty **body mass** perception object.*
- elemental real(srp) function percept_bodymass_get_current (this)

  *Get the current value of the **body mass** perception.*
- subroutine percept_bodymass_update_current (this, current)

  *Set and update the current **body mass** perception value.*
- elemental subroutine percept_bodymass_destroy_deallocate (this)

  *Destroy the **body mass** perception object and deallocate.*
- elemental subroutine percept_energy_create_init (this)

  *Initiate an empty **energy** perception object.*

- elemental real(srp) function percept_energy_get_current (this)

  *Get the current value of the **energy** reserves.*
- subroutine percept_energy_update_current (this, current)

  *Set and update the current **energy** perception value.*
- elemental subroutine percept_energy_destroy_deallocate (this)

  *Destroy the **energy** perception object and deallocate.*
- elemental subroutine percept_age_create_init (this)

  *Initiate an empty **age** perception object.*
- elemental integer function percept_age_get_current (this)

  *Get the current value of the **age** reserves.*
- subroutine percept_age_update_current (this, current)

  *Set and update the current **age** perception value.*
- elemental subroutine percept_age_destroy_deallocate (this)

  *Destroy the **age** perception object and deallocate it.*
- subroutine spatial_percept_set_cid (this, id)

  *Set unique **id** for the conspecific perception component.*
- elemental integer function spatial_percept_get_cid (this)

  *Get the unique **id** of the food item object.*
- elemental subroutine consp_percept_comp_create (this)

  Create a single conspecific perception component at an undefined position with default properties.
- subroutine consp_percept_make (this, location, size, mass, dist, cid, is_male)

  Make a single conspecific perception component. This is a single conspecific located within the visual range of the agent.
- elemental real(srp) function consp_percept_get_size (this)

  *Get the **conspecific** perception component body size.*
- elemental real(srp) function consp_percept_get_mass (this)

  *Get the **conspecific** perception component body mass.*
- elemental real(srp) function consp_percept_get_dist (this)

  *Get the **conspecific** perception component distance.*
- elemental logical function consp_percept_sex_is_male_get (this)

  *Get the **conspecific** perception component sex flag (male).*
- elemental logical function consp_percept_sex_is_female_get (this)

  *Get the **conspecific** perception component sex flag (female).*
- elemental subroutine percept_consp_create_init (this, maximum_number_conspecifics)

  Create conspecifics perception object, it is an array of conspecific perception components.
- elemental subroutine percept_consp_number_seen (this, number_set)

  *Set the total number of conspecifics perceived (seen) in the conspecific perception object. But do **not** reallocate the conspecific perception components so far.*
- pure subroutine percept_consp_make_fill_arrays (this, consps)

  Make the conspecifics perception object, fill it with the actual arrays.
- elemental integer function percept_consp_get_count_seen (this)

  Get the number (count) of conspecifics seen. Trivial.
- elemental subroutine percept_consp_destroy_deallocate (this)

  Deallocate and delete a conspecific perception object.
- subroutine consp_perception_get_visrange_objects (this, consp_agents, time_step_model)

  Get available conspecific perception objects within the visual range of the agent, which the agent can perceive and therefore respond to.
- elemental logical function consp_perception_is_seeing_conspecifics (this)

  Check if the agent sees any conspecifics within the visual range.
- elemental subroutine spatialobj_percept_comp_create (this)

  Create a single arbitrary spatial object perception component at an undefined position with default properties.

- subroutine spatialobj_percept_make (this, location, size, dist, cid)

  *Make a single arbitrary **spatial** object perception component.*
- elemental real(srp) function spatialobj_percept_get_size (this)

  Get an arbitrary spatial object perception component size.
- elemental real(srp) function spatialobj_percept_get_dist (this)

  Get the distance to an arbitrary spatial object perception component.
- real(srp) function spatialobj_percept_visibility_visual_range (this, object_area, contrast, time_step_model)

  Calculate the visibility range of this spatial object. Wrapper to the `visual_range` function. This function calculates the distance from which this object can be seen by a visual object (e.g. predator or prey).
- elemental subroutine percept_predator_create_init (this, maximum_number_predators)

  *Create **predator** perception object, it is an array of spatial perception components.*
- elemental subroutine percept_predator_number_seen (this, number_set)

  *Set the total number of **predators** perceived (seen) in the predator perception object. But do not reallocate the predator perception components so far.*
- pure subroutine percept_predator_make_fill_arrays (this, preds, attack_rate)

  *Make the **predator** perception object, fill it with the actual arrays.*
- pure subroutine percept_predator_set_attack_rate_vector (this, attack_rate)

  Set an array of the attack rates for the predator perception object.
- pure subroutine percept_predator_set_attack_rate_scalar (this, attack_rate)

  Set an array of the attack rates for the predator perception object.
- elemental integer function percept_predator_get_count_seen (this)

  Get the number (count) of predators seen. Trivial.
- elemental subroutine percept_predator_destroy_deallocate (this)

  *Deallocate and delete a **predator** perception object.*
- subroutine predator_perception_get_visrange_objects (this, spatl_agents, time_step_model)

  Get available predators perception objects within the visual range of the agent, which the agent can perceive and therefore respond to.
- elemental logical function predator_perception_is_seeing_predators (this)

  Check if the agent sees any predators within the visual range.
- elemental subroutine percept_light_create_init (this)

  Make en empty light perception component. Really necessary only when perception objects are all allocatable.
- elemental real(srp) function percept_light_get_current (this)

  Get the current perception of the illumination.
- subroutine percept_light_set_current (this, timestep, depth)

  *Set the current **light** level into the perception component.*
- elemental subroutine percept_light_destroy_deallocate (this)

  *Destroy / deallocate **light** perception component. Really necessary only when perception objects are all allocatable.*
- subroutine light_perception_get_object (this, time_step_model)

  *Get **light** perception objects into the individual PERCEPTION object layer.*
- elemental subroutine percept_depth_create_init (this)

  Make en empty depth perception component. Really necessary only when perception objects are all allocatable.
- elemental real(srp) function percept_depth_get_current (this)

  *Get the current perception of the **depth**.*
- subroutine percept_depth_set_current (this, cdepth)

  *Set the current **depth** level into the perception component.*
- elemental subroutine percept_depth_destroy_deallocate (this)

  *Destroy / deallocate **depth** perception component. Really necessary only when perception objects are all allocatable.*
- elemental subroutine percept_reprfac_create_init (this)

  Make en empty reproductive factor perception component. Really necessary only when perception objects are all allocatable.
- elemental real(srp) function percept_reprfac_get_current (this)

*Get the current perception of the **reproductive factor**.*

- subroutine percept_reprfac_set_current (this, reprfac)

  *Set the current **reproductive factor** level into perception component.*
- elemental subroutine percept_reprfac_destroy_deallocate (this)

  *Destroy / deallocate **reproductive factor** perception component. Really necessary only when perception objects are all allocatable.*
- subroutine depth_perception_get_object (this)

  *Get **depth** perception objects into the **individual** PERCEPTION object layer.*
- subroutine stomach_perception_get_object (this)

  *Get the **stomach capacity** perception objects into the **individual** PERCEPTION object layer.*
- subroutine bodymass_perception_get_object (this)

  *Get the **body mass** perception objects into the **individual** PERCEPTION object layer.*
- subroutine energy_perception_get_object (this)

  *Get the **energy reserves** perception objects into the **individual** PERCEPTION object layer.*
- subroutine age_perception_get_object (this)

  *Get the **age** perception objects into the **individual** PERCEPTION object layer.*
- subroutine repfac_perception_get_object (this)

  *Get the **reproductive factor** perception objects into the **individual** PERCEPTION object layer.*
- elemental subroutine percept_memory_add_to_stack (this, light, depth, food, foodsize, fooddist, consp, pred, stom, bdmass, energ, reprfac)

  Add perception components into the memory stack.
- elemental subroutine percept_memory_cleanup_stack (this)

  Cleanup and destroy the perceptual memory stack.
- elemental integer function percept_memory_food_get_total (this)

  Get the total number of food items within the whole perceptual memory stack.
- elemental real(srp) function percept_memory_food_get_mean_n (this, last)

  *Get the **average number** of food items per single time step within the whole perceptual memory stack.*
- elemental subroutine percept_memory_food_mean_n_split (this, window, split_val, older, newer)

  *Get the **average number** of food items per single time step within the perceptual memory stack, split to the first (older) and second (newer) parts. The whole memory stack ('sample') is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it.*
- elemental real(srp) function percept_memory_food_get_mean_size (this, last)

  *Get the **average size** of food item per single time step within the whole perceptual memory stack.*
- elemental subroutine percept_memory_food_mean_size_split (this, window, split_val, older, newer)

  *Get the **average size** of food items per single time step within the perceptual memory stack, split to the first (older) and second(newer) parts. The whole memory stack 'sample' is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it.*
- elemental real(srp) function percept_memory_food_get_mean_dist (this, last, undef_ret_null)

  *Get the **average distance** to food item per single time step within the whole perceptual memory stack.*
- elemental subroutine percept_memory_food_mean_dist_split (this, window, split_val, older, newer)

  *Get the **average distance** to food items per single time step within the perceptual memory stack, split to the first (older) and second(newer) parts. The whole memory stack 'sample' is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it.*
- elemental real(srp) function percept_memory_consp_get_mean_n (this, last)

  *Get the **average number** of conspecifics per single time step within the whole perceptual memory stack.*
- elemental integer function percept_memory_predators_get_total (this)

  Get the total number of predators within the whole perceptual memory stack.
- elemental real(srp) function percept_memory_predators_get_mean (this, last)

  Get the average number of predators per single time step within the whole perceptual memory stack.
- elemental subroutine percept_memory_predators_mean_split (this, window, split_val, older, newer)

  *Get the **average number** of predators per single time step within the perceptual memory stack, split to the first (older) and second(newer) parts. The whole memory stack ('sample') is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it.*

- elemental subroutine perception_objects_add_memory_stack (this)

  *Add the various perception objects to the memory stack object. This procedure is called **after** all the perceptual components (light, depth food, conspecifics, predators, etc.) are collected (using `set` object-bound subroutines) into the perception bundle, so all the values are known and ready to be used.*
- subroutine perception_objects_get_all_environmental (this)

  *A single umbrella subroutine to get all **environmental** perceptions: light, depth. This procedure invokes these calls:*
- subroutine perception_objects_get_all_inner (this)

  *A single umbrella subroutine wrapper to get all **inner** perceptions: stomach, body mass, energy, age. Invokes all these procedures:*
- elemental subroutine, private perception_objects_init_agent (this)

  Initialise all the perception objects for the current agent. Do not fill perception objects with the real data yet.
- elemental subroutine perception_objects_destroy (this, clean_memory)

  Destroy and deallocate all perception objects.
- elemental real(srp) function perception_predation_risk_objective (this)

  *Calculate the risk of **predation** as being **perceived** / **assessed** by this agent.*
- elemental real(srp) function predation_risk_backend (pred_count, pred_memory_mean, weight_direct)

  Simple computational backend for the risk of predation that is used in objective risk function the_neurobio::perception_predation_risk_obje and the subjective risk function.
- elemental subroutine perception_components_attention_weights_init (this, all_vals_fix, all_one, weight_↩ light, weight_depth, weight_food_dir, weight_food_mem, weight_conspec, weight_pred_dir, weight_predator, weight_stomach, weight_bodymass, weight_energy, weight_age, weight_reprfac)

  Initialise the attention components of the emotional state to their default parameter values. Attention sets weights to individual perceptual components when the overall weighted sum is calculated. The default weights are parameters defined in `COMMONDATA`.
- subroutine perception_components_neuronal_response_init_set (this, this_agent, param_gp_matrix_light, param_gp_matrix_depth, param_gp_matrix_food_dir, param_gp_matrix_food_mem, param_gp_matrix_↩ conspec, param_gp_matrix_pred_dir, param_gp_matrix_predator, param_gp_matrix_stomach, param_gp↩ _matrix_bodymass, param_gp_matrix_energy, param_gp_matrix_age, param_gp_matrix_reprfac, param_↩ gerror_cv_light, param_gerror_cv_depth, param_gerror_cv_food_dir, param_gerror_cv_food_mem, param↩ _gerror_cv_conspec, param_gerror_cv_pred_dir, param_gerror_cv_predator, param_gerror_cv_stomach, param_gerror_cv_bodymass, param_gerror_cv_energy, param_gerror_cv_age, param_gerror_cv_reprfac, param_gene_label_light, param_gene_label_depth, param_gene_label_food_dir, param_gene_label_food↩ _mem, param_gene_label_conspec, param_gene_label_pred_dir, param_gene_label_predator, param_↩ gene_label_stomach, param_gene_label_bodymass, param_gene_label_energy, param_gene_label_age, param_gene_label_reprfac)

  *Set and calculate individual perceptual components for **this** motivational state using the **neuronal response** function, for **this_agent**.*
- subroutine perception_components_neuronal_response_calculate (this, this_agent, param_gp_matrix_light, param_gp_matrix_depth, param_gp_matrix_food_dir, param_gp_matrix_food_mem, param_gp_matrix_↩ conspec, param_gp_matrix_pred_dir, param_gp_matrix_predator, param_gp_matrix_stomach, param_gp↩ _matrix_bodymass, param_gp_matrix_energy, param_gp_matrix_age, param_gp_matrix_reprfac, param_↩ gerror_cv_light, param_gerror_cv_depth, param_gerror_cv_food_dir, param_gerror_cv_food_mem, param↩ _gerror_cv_conspec, param_gerror_cv_pred_dir, param_gerror_cv_predator, param_gerror_cv_stomach, param_gerror_cv_bodymass, param_gerror_cv_energy, param_gerror_cv_age, param_gerror_cv_reprfac, perception_override_light, perception_override_depth, perception_override_food_dir, perception_override↩ _food_mem, perception_override_conspec, perception_override_pred_dir, perception_override_predator, perception_override_stomach, perception_override_bodymass, perception_override_energy, perception_↩ override_age, perception_override_reprfac)

  *Calculate individual perceptual components for **this** motivational state using the **neuronal response** function, for an **this_agent**.*
- elemental real(srp) function state_motivation_light_get (this)

  *Standard "get" function for the state neuronal **light** effect component.*
- elemental real(srp) function state_motivation_depth_get (this)

  *Standard "get" function for the state neuronal **depth** effect component.*
- elemental real(srp) function state_motivation_food_dir_get (this)

*Standard "get" function for the state neuronal **directly seen food** effect component.*

- elemental real(srp) function state_motivation_food_mem_get (this)

  *Standard "get" function for the state neuronal **food items from past memory** effect component.*
- elemental real(srp) function state_motivation_conspec_get (this)

  *Standard "get" function for the state neuronal **conspecifics** effect component.*
- elemental real(srp) function state_motivation_pred_dir_get (this)

  *Standard "get" function for the state neuronal **direct predation** effect component.*
- elemental real(srp) function state_motivation_predator_get (this)

  *Standard "get" function for the state neuronal **predators** effect component.*
- elemental real(srp) function state_motivation_stomach_get (this)

  *Standard "get" function for the state neuronal **stomach** effect component.*
- elemental real(srp) function state_motivation_bodymass_get (this)

  *Standard "get" function for the state neuronal **body mass** effect component.*
- elemental real(srp) function state_motivation_energy_get (this)

  *Standard "get" function for the state neuronal **energy reserves** effect component.*
- elemental real(srp) function state_motivation_age_get (this)

  *Standard "get" function for the state neuronal **age** effect component.*
- elemental real(srp) function state_motivation_reprfac_get (this)

  *Standard "get" function for the state neuronal **reproductive factor** effect component.*
- elemental real(srp) function state_motivation_motivation_prim_get (this)

  *Standard "get" function for the root state, get the overall **primary motivation value** (before modulation).*
- elemental real(srp) function state_motivation_motivation_get (this)

  *Standard "get" function for the root state, get the overall **final motivation value** (after modulation).*
- elemental logical function state_motivation_is_dominant_get (this)

  Check if the root state is the dominant state in GOS.
- elemental character(len=label_length) function state_motivation_fixed_label_get (this)

  Get the fixed label for this motivational state. Note that the label is fixed and cannot be changed.
- pure subroutine state_motivation_attention_weights_transfer (this, copy_from)

  Transfer attention weights between two motivation state components. The main use of this subroutine would be to transfer attention from the actor agent's main motivation's attention component to the behaviour's GOS expectancy object.
- elemental real(srp) function perception_component_maxval (this)

  *Calculate the **maximum** value over all the perceptual components.*
- elemental real(srp) function state_motivation_percept_maxval (this)

  *Calculate the **maximum** value over all the perceptual components of this motivational state component.*
- elemental real(srp) function state_motivation_calculate_prim (this, maxvalue)

  *Calculate the level of **primary motivation** for this **specific** emotional state **component**.*
- elemental subroutine perception_component_motivation_init_zero (this)

  Initialise perception components for a motivation state object.
- elemental subroutine state_hunger_zero (this)

  *Init and cleanup **hunger** motivation object. The only difference from the base root STATE_MOTIVATION_BASE is that it sets unique label.*
- elemental subroutine state_fear_defence_zero (this)

  *Init and cleanup **fear state** motivation object. The only difference from the base root STATE_MOTIVATION_BASE is that it sets unique label.*
- elemental subroutine state_reproduce_zero (this)

  *Init and cleanup **reproductive** motivation object. The only difference from the base root STATE_MOTIVATION_BASE is that it sets unique label.*
- elemental subroutine motivation_init_all_zero (this)

  Init the expectancy components to a zero state.
- elemental subroutine motivation_reset_gos_indicators (this)

  Reset all GOS indicators for this motivation object.

- elemental real(srp) function motivation_max_perception_calc (this)

  Calculate maximum value of the perception components across all motivations.
- pure real(srp) function, dimension(:), allocatable motivation_return_final_as_vector (this)

  Return the vector of final motivation values for all motivational state components.
- elemental real(srp) function motivation_maximum_value_motivation_finl (this)

  Calculate the maximum value of the final motivations across all motivational state components.
- elemental logical function motivation_val_is_maximum_value_motivation_finl (this, test_value)

  *Checks if the test value is the maximum **final** motivation value across all motivational state components.*
- elemental logical function motivation_val_is_maximum_value_motivation_finl_o (this, test_motivation)

  *Checks if the test value is the maximum **final** motivation value across all motivational state components.*
- elemental subroutine motivation_primary_sum_components (this, max_val)

  *Calculate the **primary motivations** from motivation-specific perception appraisal components. The **primary motivations** are motivation values before the modulation takes place.*
- elemental subroutine motivation_modulation_absent (this)

  *Produce **modulation** of the primary motivations, that result in the **final motivation** values (`_finl`). In this subroutine, **modulation is absent**, so the final motivation values are equal to the primary motivations.*
- elemental subroutine, private appraisal_init_zero_cleanup_all (this)

  Initialise and cleanup all appraisal object components and sub-objects.
- elemental subroutine appraisal_agent_set_dead (this)

  *Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function.*
- subroutine appraisal_perceptual_comps_motiv_neur_response_calculate (this)

  Get the perceptual components of all motivational states by passing perceptions via the neuronal response function.
- elemental subroutine appraisal_primary_motivations_calculate (this, rescale_max_motivation)

  Calculate primary motivations from perceptual components of each motivation state.
- subroutine appraisal_motivation_modulation_non_genetic (this, no_modulation)

  *Produce **modulation** of the primary motivations, that result in the **final motivation** values (`_finl`). Modulation here is non-genetic and involves a fixed transformation of the primary motivation values.*
- subroutine appraisal_motivation_modulation_genetic (this, no_genetic_modulation)

  *Produce **modulation** of the primary motivations, that result in the **final motivation** values (`_finl`). Modulation involves effects of such characteristics of the agent as body mass and age on the primary motivations (hunger, active and passive avoidance and reproduction) mediated by the genome effects. Here the genome determines the coefficients that set the degree of the influence of the agent's characteristics on the motivations.*
- elemental subroutine appraisal_add_final_motivations_memory (this)

  Add individual final emotional state components into the emotional memory stack. This is a wrapper to the the_neurobio::memory_emotional::add_to_memory method.
- real(srp) function reproduce_do_probability_reproduction_calc (this, weight_baseline, allow_immature)

  Calculate the instantaneous probability of successful reproduction.
- logical function reproduction_success_stochast (this, prob)

  *Determine a stochastic outcome of **this** agent reproduction. Returns TRUE if the agent has reproduced successfully.*
- elemental subroutine emotional_memory_add_to_stack (this, v_hunger, v_defence_fear, v_reproduction, v↩ _gos_label, v_gos_arousal, v_gos_repeated)

  Add emotional components into the memory stack.
- elemental subroutine emotional_memory_add_gos_to_stack (this, v_gos_label, v_gos_arousal, v_gos_↩ repeated)

  Add the current GOS label or/and arousal value and/or arousal repeat count into the emotional memory stack.
- elemental subroutine emotional_memory_cleanup_stack (this)

  Cleanup and destroy the emotional memory stack.
- elemental real(srp) function emotional_memory_hunger_get_mean (this, last)

  Get the average value of the hunger motivation state within the whole emotional memory stack.
- elemental real(srp) function emotional_memory_actve_avoid_get_mean (this, last)

  Get the average value of the fear state motivation state within the whole emotional memory stack.

- elemental real(srp) function emotional_memory_reproduct_get_mean (this, last)

  Get the average value of the reproductive motivation state within the whole emotional memory stack.
- elemental real(srp) function emotional_memory_arousal_mean (this, last)

  Get the average value of the GOS arousal within the whole emotional memory stack.
- subroutine gos_find_global_state (this)

  *Find and set the **Global Organismic State (GOS)** of the agent based on the various available motivation values. The motivation values linked with the different stimuli compete with the current GOS and among themselves.*
- elemental subroutine, private gos_init_zero_state (this)

  Initialise GOS engine components to a zero state. The values are set to commondata::missing, commondata::unknown, string to "undefined".
- elemental subroutine gos_agent_set_dead (this)

  *Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function.*
- elemental subroutine gos_reset_motivations_non_dominant (this)

  Reset all motivation states as not dominant with respect to the GOS.
- elemental character(len=label_length) function gos_global_get_label (this)

  Get the current global organismic state (GOS).
- elemental real(srp) function gos_get_arousal_level (this)

  Get the overall level of arousal. Arousal is the current level of the dominant motivation that has brought about the current GOS at the previous time step.
- subroutine gos_attention_modulate_weights (this)

  Modulate the attention weights to suppress all perceptions alternative to the current GOS. This is done using the attention modulation interpolation curve.
- elemental integer function perception_food_items_below_calculate (this)

  *Calculate the number of food items in the perception object that are located **below** the actor agent.*
- elemental integer function perception_food_items_below_horiz_calculate (this, hz_lower, hz_upper)

  *Calculate the number of food items in the perception object that are located **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z+hz_lower, z+hz_upper].*
- elemental real(srp) function perception_food_mass_below_calculate (this)

  *Calculate the average mass of a food item from all the items in the current perception object that are **below** the actor agent.*
- elemental real(srp) function perception_food_mass_below_horiz_calculate (this, hz_lower, hz_upper)

  *Calculate the average mass of a food item from all the items in the current perception object that are **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z+hz_lower, z+hz_upper].*
- elemental integer function perception_food_items_above_calculate (this)

  *Calculate the number of food items in the perception object that are located **above** the actor agent.*
- elemental integer function perception_food_items_above_horiz_calculate (this, hz_lower, hz_upper)

  *Calculate the number of food items in the perception object that are located **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z-hz_upper, z-hz_upper].*
- elemental real(srp) function perception_food_mass_above_calculate (this)

  *Calculate the average mass of a food item from all the items in the current perception object that are **above** the actor agent.*
- elemental real(srp) function perception_food_mass_above_horiz_calculate (this, hz_lower, hz_upper)

  *Calculate the average mass of a food item from all the items in the current perception object that are **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z-hz_upper, z-hz_upper].*
- elemental integer function perception_conspecifics_below_calculate (this)

  *Calculate the number of conspecifics in the perception object that are located **below** the actor agent.*
- elemental integer function perception_conspecifics_above_calculate (this)

  *Calculate the number of conspecifics in the perception object that are located **above** the actor agent.*
- elemental integer function perception_conspecifics_below_horiz_calculate (this, hz_lower, hz_upper)

*Calculate the number of conspecifics in the perception object that are located **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z+hz_lower, z+hz_upper].*

- elemental integer function perception_conspecifics_above_horiz_calculate (this, hz_lower, hz_upper)

  *Calculate the number of conspecifics in the perception object that are located **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z-hz_upper, z-hz_upper].*
- elemental integer function perception_predator_below_calculate (this)

  *Calculate the number of predators in the perception object that are located **below** the actor agent.*
- elemental integer function perception_predator_above_calculate (this)

  *Calculate the number of predators in the perception object that are located **above** the actor agent.*
- elemental integer function perception_predator_below_horiz_calculate (this, hz_lower, hz_upper)

  *Calculate the number of predators in the perception object that are located **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z+hz_lower, z+hz_upper].*
- elemental integer function perception_predator_above_horiz_calculate (this, hz_lower, hz_upper)

  *Calculate the number of predators in the perception object that are located **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z-hz_upper, z-hz_upper].*
- elemental real(srp) function perception_food_dist_below_calculate (this)

  *Calculate the average distance to all food items in the current perception object that are **below** the actor agent.*
- elemental real(srp) function perception_food_dist_above_calculate (this)

  *Calculate the average distance to all food items in the current perception object that are **above** the actor agent.*
- elemental real(srp) function perception_consp_dist_below_calculate (this)

  *Calculate the average distance to all conspecifics in the current perception object that are **below** the actor agent.*
- elemental real(srp) function perception_consp_dist_above_calculate (this)

  *Calculate the average distance to all conspecifics in the current perception object that are **above** the actor agent.*
- elemental real(srp) function perception_predator_dist_below_calculate (this)

  *Calculate the average distance to all predators in the current perception object that are **below** the actor agent.*
- elemental real(srp) function perception_predator_dist_above_calculate (this)

  *Calculate the average distance to all predators in the current perception object that are **above** the actor agent.*
- real(srp) function predator_capture_probability_calculate_spatobj (this, this_predator, attack_rate, is_↩ freezing, time_step_model)

  Calculate the probability of attack and capture of the `this` agent by the predator `this_predator`. This probability is a function of the distance between the predator and the agent and is calculated by the predator-class-bound procedure the_environment::predator::risk_fish(). Example call:
- real(srp) function predator_capture_probability_calculate_pred (this, this_predator, is_freezing, time_step_↩ model)

  Calculate the probability of attack and capture of the `this` agent by the predator `this_predator`. This probability is a function of the distance between the predator and the agent and is calculated by the predator-class-bound procedure the_environment::predator::risk_fish().
- real(srp) function predation_capture_probability_risk_wrapper (this, is_freezing)

  Calculate the overall direct predation risk for the agent, i.e. the probability of attack and capture by the nearest predator.
- elemental real(srp) function get_prop_size (this)

  Get the body size property of a polymorphic object. The object can be of the following extension of the basic the_environment::spatial class:
- elemental real(srp) function get_prop_mass (this)

  Get the body mass property of a polymorphic object. The object can be of the following extension of the basic the_environment::spatial class:

Variables

- character(len= ∗), parameter, private modname = "(THE_NEUROBIO)"

8.10.1 Detailed Description

Definition of the decision making and behavioural the architecture.

8.10.2 THE_BEHAVIOUR module

This module defines the neurobiological architecture of the agent, starting from perception to representation, appraisal, motivation, emotion, determination of the global organismic state, and behaviour.

8.10.3 Function/Subroutine Documentation

8.10.3.1 percept_food_create_init()

```
elemental subroutine the_neurobio::percept_food_create_init (
            class(percept_food), intent(inout) this,
            integer, intent(in) maximum_number_food_items )
```

Initiate an empty **food** perception object with known number of components.

Parameters

in	*maximum_number_food_items*	maximum_number_food_items Maximum number of food items in the food perception object, normally equal to the partial food resource indexing order commondata::food_select_items_index_partial.

8.10.3.1.1 Implementation details Create all food items in the perception array (create is elemental procedure).
Initialise all other components of the perception object.
array
Set the initial number of food items in the perception object to the maximum number using the function percept↩_food_number_seen below.
Definition at line 1367 of file m_neuro.f90.

8.10.3.2 percept_food_number_seen()

```
subroutine the_neurobio::percept_food_number_seen (
            class(percept_food), intent(inout) this,
            integer, intent(in) number_set )
```

Set the total number of food items perceived (seen) in the food perception object. Do not reallocate the perception object components with respect to this new number yet.

Parameters

in	*number_set*	Set the number of food items in the perception object.

Definition at line 1409 of file m_neuro.f90.

8.10.3.3 percept_food_make_fill_arrays()

```
subroutine the_neurobio::percept_food_make_fill_arrays (
            class(percept_food), intent(inout) this,
            type(food_item), dimension(:), intent(in) items,
            real(srp), dimension(:), intent(in) dist )
```

Make the food perception object, fill it with the actual data arrays.

Note

Note that the size and allocation is set by the `init` method.

Parameters

in	*items*	items an array of food items that form the perception object.
in	*dist*	dist an array of the distances between the agent and each of the food items in the perception object.

8.10.3.3.1 Implementation details First we check for non-conforming input arrays and re-init and reallocate the perception object, if needed, to the minimum value.
Report this issue to the log.
Second, fill the dynamic food perception object with the data from the input arrays. They should have conforming sizes now.
Definition at line 1421 of file m_neuro.f90.

8.10.3.4 percept_food_get_count_found()

```
elemental integer function the_neurobio::percept_food_get_count_found (
            class(percept_food), intent(in) this )
```

Get the number (count) of food items seen. Trivial.
Definition at line 1460 of file m_neuro.f90.

8.10.3.5 percept_food_get_meansize_found()

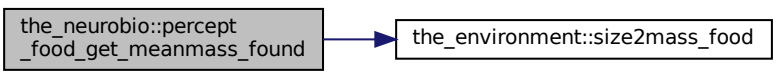
```
elemental real(srp) function the_neurobio::percept_food_get_meansize_found (
            class(percept_food), intent(in) this )
```

Get the average size of food items seen. Trivial.
Definition at line 1470 of file m_neuro.f90.

8.10.3.6 percept_food_get_meanmass_found()

```
elemental real(srp) function the_neurobio::percept_food_get_meanmass_found (
            class(percept_food), intent(in) this )
```

Get the average mass of food items seen. Trivial.
Definition at line 1484 of file m_neuro.f90.
Here is the call graph for this function:



8.10.3.7 percept_food_get_meandist_found()

```
elemental real(srp) function the_neurobio::percept_food_get_meandist_found (
            class(percept_food), intent(in) this )
```

Get the average distance to the food items seen. Trivial.
If no food items seen, we have undefined distance.
Definition at line 1500 of file m_neuro.f90.

8.10.3.8 percept_food_destroy_deallocate()

```
elemental subroutine the_neurobio::percept_food_destroy_deallocate (
            class(percept_food), intent(inout) this )
```

Deallocate and delete a **food** perception object.
Definition at line 1514 of file m_neuro.f90.

8.10.3.9 food_perception_get_visrange_objects()

```
subroutine the_neurobio::food_perception_get_visrange_objects (
            class(perception), intent(inout) this,
            class(food_resource), intent(in) food_resource_available,
            integer, intent(in), optional time_step_model )
```

Get available food items within the visual range of the agent, which the agent can perceive and therefore respond to. Food perception is packaged into the food perception object this%perceive_food for output.
Food perception is quite complex to implement as it requires determining individual food items within the current visual range of the agent. There are, however, potentially thousands (or millions) of food items in the food resource, each of the food items is stochastic (e.g. they have different sizes), so visual range differ for each item and each agent should determine food items in its proximity at numerous time steps of the model. This means repeating huge loops many times for each agent at each time step. This is approached by array segmentation: the perception object is obtained by *partial indexing* of a very limited number (=`commondata::food_select_items_index_partial`) of only the nearest food items, the agent's visual range is then determined for each of this nearest neighbouring food items, and finally those food items that individually fall within the visual range are included into the perception object.

Note

Note that there are three similar procedures that detect spatial objects within the visual range of the agent:

- the_neurobio::perception::see_food – perception of food items;
- the_neurobio::perception::see_consp – perception of conspecifics:
- the_neurobio::perception::see_pred – perception of predators.

All these procedures were actually implemented using the first (the_neurobio::perception::see_food) as a template. All three implement partial indexing of the nearest spatial objects to accelerate computation of large arrays of spatial objects.

Parameters

in	*food_resource_available*	food_resource_available Global food resource object from which we select neighbouring item components that are present within the visual range of the agent.
in	*time_step_model*	time_step_model The current time step of the model.

8.10.3.9.1 Notable variables and parameters

- **dist_foods** - temporary array of food items (`the_environment::food_item`) available to the agent.

dist_food_neighbours - temporary array of the distances to the neighbouring food items.

- **dist_food_index** - temporary partial index vector for the distances to the neighbouring food items.
- **irradiance_agent_depth** - local variable defining the irradiance (illumination) at the current depth of the agent. Needed to calculate the agent's visual range.
- **food_item_area** - local variable defining the area of the food item. It is an array, area of each item in the `food_resource_available` and `dist_foods`. Needed to calculate the agent's visual range.
- **food_item_visual_range** - local variable defining the visual range of the agent for detecting each of the food items (with known areas) at the agent's current depth.
- **food_items_percept_in_visrange** - local sorted array of food objects that are within the visual range of the agent for output. The array should normally have the size of commondata::food_select_items_index_partial elements, but only the first `food_items_n_visrange` elements of it are actually within the visual range.
- **food_items_dist_sorted** - temporary local sorted array of distances between the agent and each of the nearest neighbouring food items, sorted for output.
- **food_items_n_visrange** - local number of elements of `food_items_percept_in_visrange` for output that are within he visual range of the agent.

8.10.3.9.2 Implementation details

8.10.3.9.2.1 Checks and preparations Check optional time step parameter. If unset, use global `commondata::global_time_step_model_current`.
Initialise index and rank values. Unitit ialised index arrays may result in invalid memory reference in `ARRAY_INDEX` (it is not safe by design).
Copy food items array component from the `food_resource_available` class' the_environment::food_item's array.

Warning

Note that we cannot here call

```
call dist_foods%position( food_resource_available%food%location() )
```

as the objects are `the_environment::food_item` higher level than `the_environment::spatial`.

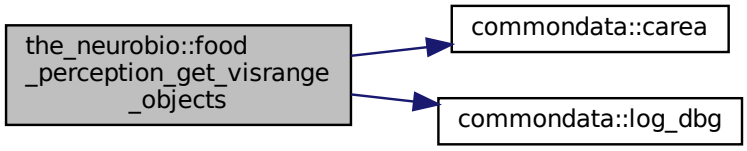
8.10.3.9.2.2 Step 1 First, we determine up to the maximum order (fast *partial indexing*) of `commondata::food_select_iter` neighbouring food items that are in proximity of the agent. This is done using the the_environment::spatial::neighbours() backend procedure.

8.10.3.9.2.3 Step 2 Second, we select only those items within this set, which are within the visual range of the agent under the current conditions. To do this we, first, calculate the ambient illumination/irradiance level at the depth of the agent. Done using the the_environment::spatial::illumination() procedure.
Compute the array of "prey areas" for each of the food items (whole array or neighbouring food items only).
Compute the vector of the visual ranges for detecting each of the food items by the agent.

8.10.3.9.2.4 Step 3 Now we can get the pre-output array `food_items_percept_in_visrange` that contains the food objects available within the visual range of the agent. Also, we count the number of such items for the output parameter `food_items_n_visrange`.
Also, check if the food item is available (not eaten).
Here we also log warning if no food items found, when debugging (see commondata::is_debug).

8.10.3.9.2.5 Step 4 Finally, we can now create the output food perception object, including only food items that are within the current visual range of the agent. Init (create and allocate) the food perception object (at first empty) using the the_neurobio::percept_food::init().
Fill the output perception object with the values obtained at the step 3. This is done using the the_neurobio::percept_food::make() backend procedure.
Definition at line 1553 of file m_neuro.f90.

Here is the call graph for this function:



8.10.3.10 food_perception_is_seeing_food()

```
elemental logical function the_neurobio::food_perception_is_seeing_food (
            class(perception), intent(in) this )
```

Check if the agent sees any food items within its visual range.

Warning

Should be called after the `see_food` method as it is only an accessor get-function.

Returns

Returns TRUE if the agent has any food items in its perception object and FALSE otherwise.

Definition at line 1765 of file m_neuro.f90.

8.10.3.11 food_perception_probability_capture_memory_object()

```
real(srp) function the_neurobio::food_perception_probability_capture_memory_object (
            class(perception), intent(in) this,
            integer, intent(in), optional last,
            integer, intent(in), optional time_step_model )
```

Calculate the probability of capture of a subjective representation of food item based on the data from the perceptual memory stack.

Parameters

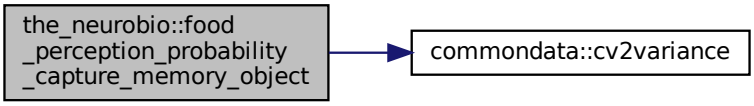
in	*last*	last Limit to only this number of latest components in history.
in	*time_step_model*	time_step_model optional time step of the model, if absent, obtained from the global variable `commondata::global_time_step_model_current`.

Returns

Capture probability of the "subjective" food item that has the size equal to the size of the average memorised food items (from the agent's perception memory stack) and located at an average distance of food items from the memory stack.

8.10.3.11.1 Implementation notes `subjective_food_item_average` of the type the_environment::food_item is a subjective representation of the food item object built from the memory stack data.
First, check optional time step parameter. If unset, use global commondata::global_time_step_model_current.
Second, build the subjective food item `subjective_food_item_average` using the the_environment::↩
: food_item::make() method. The location of this subjective food item coincides with the location of the agent.

This allows to calculate the visibility (visual range) of the food items bat the depth of the agent.
Then the capture probability is calculated using the type-bound method the_environment::food_item::capture_probability().
Importantly, the distance towards towards the food item is explicitly provided as the average distance from the memory stack calculated by the the_neurobio::memory_perceptual::get_food_mean_dist().
Finally, we add a random Gaussian error to the above objective value. Now we have obtained the stochastic subjective value of the capture probability for this food item including a Gaussian error. There is also a strong limitation for the subjective probability to be within the range [0.0, 1.0]. See also subjective_capture_prob() for a similar Gaussian error in subjective probability.
Definition at line 1779 of file m_neuro.f90.
Here is the call graph for this function:



8.10.3.12 percept_stomach_create_init()

```
elemental subroutine the_neurobio::percept_stomach_create_init (
            class(percept_stomach), intent(inout) this )
```

Initiate an empty **stomach** capacity perception object.
First, assign the current stomach capacity to MISSING.
Definition at line 1858 of file m_neuro.f90.

8.10.3.13 percept_stomach_get_avail_capacity()

```
elemental real(srp) function the_neurobio::percept_stomach_get_avail_capacity (
            class(percept_stomach), intent(in) this )
```

Get the currently available value of the available **stomach** volume.

Returns

the stomach capacity currently available for new food.

Definition at line 1868 of file m_neuro.f90.

8.10.3.14 percept_stomach_update_avail_capacity()

```
subroutine the_neurobio::percept_stomach_update_avail_capacity (
            class(percept_stomach), intent(inout) this,
            real(srp), intent(in) current_volume )
```

Set and update the currently available value of the available **stomach** volume.

Parameters

in	*current_volume*	current_volume the new (updated) current volume of the stomach capacity.

Definition at line 1882 of file m_neuro.f90.

8.10.3.15 percept_stomach_destroy_deallocate()

```
elemental subroutine the_neurobio::percept_stomach_destroy_deallocate (
            class(percept_stomach), intent(inout) this )
```

Destroy the **stomach** perception object and deallocate it.
Set the current value to commondata::missing.
Definition at line 1896 of file m_neuro.f90.

8.10.3.16 percept_bodymass_create_init()

```
elemental subroutine the_neurobio::percept_bodymass_create_init (
            class(percept_body_mass), intent(inout) this )
```

Initiate an empty **body mass** perception object.
Assign the current body mass to commondata::missing.
Definition at line 1910 of file m_neuro.f90.

8.10.3.17 percept_bodymass_get_current()

```
elemental real(srp) function the_neurobio::percept_bodymass_get_current (
            class(percept_body_mass), intent(in) this )
```

Get the current value of the **body mass** perception.

Returns

the current body mass value.

Definition at line 1920 of file m_neuro.f90.

8.10.3.18 percept_bodymass_update_current()

```
subroutine the_neurobio::percept_bodymass_update_current (
            class(percept_body_mass), intent(inout) this,
            real(srp), intent(in) current )
```

Set and update the current **body mass** perception value.

Parameters

in	*current*	current the new (updated) current volume of the stomach capacity.

Definition at line 1933 of file m_neuro.f90.

8.10.3.19 percept_bodymass_destroy_deallocate()

```
elemental subroutine the_neurobio::percept_bodymass_destroy_deallocate (
            class(percept_body_mass), intent(inout) this )
```

Destroy the **body mass** perception object and deallocate.
Set the current value to commondata::missing.
Definition at line 1947 of file m_neuro.f90.

8.10.3.20 percept_energy_create_init()

```
elemental subroutine the_neurobio::percept_energy_create_init (
            class(percept_energy), intent(inout) this )
```

Initiate an empty **energy** perception object.
Assign the current energy to commondata::missing.

Definition at line 1961 of file m_neuro.f90.

8.10.3.21 percept_energy_get_current()

```
elemental real(srp) function the_neurobio::percept_energy_get_current (
            class(percept_energy), intent(in) this )
```

Get the current value of the **energy** reserves.

Returns

the current energy reserve.

Definition at line 1971 of file m_neuro.f90.

8.10.3.22 percept_energy_update_current()

```
subroutine the_neurobio::percept_energy_update_current (
            class(percept_energy), intent(inout) this,
            real(srp), intent(in) current )
```

Set and update the current **energy** perception value.

Parameters

in	*current*	current the new (updated) current energy reserves.

Definition at line 1984 of file m_neuro.f90.

8.10.3.23 percept_energy_destroy_deallocate()

```
elemental subroutine the_neurobio::percept_energy_destroy_deallocate (
            class(percept_energy), intent(inout) this )
```

Destroy the **energy** perception object and deallocate.
Definition at line 1997 of file m_neuro.f90.

8.10.3.24 percept_age_create_init()

```
elemental subroutine the_neurobio::percept_age_create_init (
            class(percept_age), intent(inout) this )
```

Initiate an empty **age** perception object.
Assign the current age to commondata::unknown.
Definition at line 2011 of file m_neuro.f90.

8.10.3.25 percept_age_get_current()

```
elemental integer function the_neurobio::percept_age_get_current (
            class(percept_age), intent(in) this )
```

Get the current value of the **age** reserves.

Returns

the current age.

Definition at line 2021 of file m_neuro.f90.

8.10.3.26 percept_age_update_current()

```
subroutine the_neurobio::percept_age_update_current (
            class(percept_age), intent(inout) this,
            integer, intent(in) current )
```

Set and update the current **age** perception value.

Parameters

in	*current*	current the new (updated) current age.

Definition at line 2034 of file m_neuro.f90.

8.10.3.27 percept_age_destroy_deallocate()

```
elemental subroutine the_neurobio::percept_age_destroy_deallocate (
            class(percept_age), intent(inout) this )
```

Destroy the **age** perception object and deallocate it.
Set the current value to commondata::unknown.
Definition at line 2047 of file m_neuro.f90.

8.10.3.28 spatial_percept_set_cid()

```
subroutine the_neurobio::spatial_percept_set_cid (
            class(spatial_percept_component), intent(inout) this,
            integer, intent(in), optional id )
```

Set unique **id** for the conspecific perception component.

Parameters

in	*id*	iid optional individual id number for the food item.

8.10.3.28.1 Implementation details

HUGE_ID is a local parameter, the maximum unique id ever possible.
Check if conspecific cid is provided and if not, set random within the huge range.
Definition at line 2061 of file m_neuro.f90.

8.10.3.29 spatial_percept_get_cid()

```
elemental integer function the_neurobio::spatial_percept_get_cid (
            class(spatial_percept_component), intent(in) this )
```

Get the unique **id** of the food item object.

Returns

cid the individual id number of this perception component.

Definition at line 2083 of file m_neuro.f90.

8.10.3.30 consp_percept_comp_create()

```
elemental subroutine the_neurobio::consp_percept_comp_create (
            class(conspec_percept_comp), intent(inout) this )
```

Create a single conspecific perception component at an undefined position with default properties.

8.10.3.30.1 Implementation details We here just set an undefined location of the food object using standard interface function `missing`.
Set cid to UNKNOWN.

Warning

random id on create is now disabled to allow elemental function, because random are never pure. So care to set cid's elsewhere.

Then we set the conspecific size. Should it be MISSING or grand average? thisconsp_body_size = MISSING
Set the conspecific mass. The default mass of the conspecific is twice the minimum body mass. There is no upper limit on the body mass.

Note

These values are not very important as they are for default init only and will be overwritten by the actual values.

Init distance towards the conspecific, now with MISSING value.
Init the sex is male by default, it is arbitrary.
Definition at line 2100 of file m_neuro.f90.

8.10.3.31 consp_percept_make()

```
subroutine the_neurobio::consp_percept_make (
             class(conspec_percept_comp), intent(inout) this,
             type(spatial), intent(in) location,
             real(srp), intent(in), optional size,
             real(srp), intent(in), optional mass,
             real(srp), intent(in), optional dist,
             integer, intent(in), optional cid,
             logical, intent(in), optional is_male )
```

Make a single conspecific perception component. This is a single conspecific located within the visual range of the agent.

Parameters

in	*location*	Location of the conspecific perception component, as a `SPATIAL` type container
in	*size*	size This is the optional conspecific body size as guessed by the agent. May or may not reflect the "true" size of the conspecific.
in	*mass*	mass This is the optional conspecific body mass as guessed by the agent. May or may not reflect the "true" mass of the conspecific.
in	*dist*	dist The distance towards this conspecific.
in	*cid*	iid Optional cid for the conspecific perception component. If not provided, set random.
in	*is_male*	is_male Optional flag that sex is male.

8.10.3.31.1 Implementation details We here just set the location of the food object using standard interface function `position`.
If individual id is provided, set it. If not, set random.
Then we set the conspecific perception component body size. Check if optional size is provided and left untouched if not.
Then we set the conspecific perception component body mass. Check if optional size is provided and left untouched if not.
Also set the distance towards the conspecific if provided. If not provided, ignore.
Definition at line 2134 of file m_neuro.f90.

8.10.3.32 consp_percept_get_size()

```
elemental real(srp) function the_neurobio::consp_percept_get_size (
            class(conspec_percept_comp), intent(in) this )
```

Get the **conspecific** perception component body size.
Definition at line 2199 of file m_neuro.f90.

8.10.3.33 consp_percept_get_mass()

```
elemental real(srp) function the_neurobio::consp_percept_get_mass (
            class(conspec_percept_comp), intent(in) this )
```

Get the **conspecific** perception component body mass.
Definition at line 2209 of file m_neuro.f90.

8.10.3.34 consp_percept_get_dist()

```
elemental real(srp) function the_neurobio::consp_percept_get_dist (
            class(conspec_percept_comp), intent(in) this )
```

Get the **conspecific** perception component distance.
Definition at line 2220 of file m_neuro.f90.

8.10.3.35 consp_percept_sex_is_male_get()

```
elemental logical function the_neurobio::consp_percept_sex_is_male_get (
            class(conspec_percept_comp), intent(in) this )
```

Get the **conspecific** perception component sex flag (male).
Definition at line 2231 of file m_neuro.f90.

8.10.3.36 consp_percept_sex_is_female_get()

```
elemental logical function the_neurobio::consp_percept_sex_is_female_get (
            class(conspec_percept_comp), intent(in) this )
```

Get the **conspecific** perception component sex flag (female).
Definition at line 2241 of file m_neuro.f90.

8.10.3.37 percept_consp_create_init()

```
elemental subroutine the_neurobio::percept_consp_create_init (
            class(percept_conspecifics), intent(inout) this,
            integer, intent(in) maximum_number_conspecifics )
```

Create conspecifics perception object, it is an array of conspecific perception components.

Parameters

in	*maximum_number_conspecifics*	maximum_number_conspecifics The maximum number of conspecifics in the conspecifics perception object. Normally equal to the partial conspecific selection indexing order CONSP_SELECT_ITEMS_INDEX_PARTIAL.

8.10.3.37.1 Implementation details Allocate the array of the conspecific perception components
And create all perception components (create is `elemental`).
Set the initial number of conspecifics within the visual range to the maximum number provided.
Definition at line 2256 of file m_neuro.f90.

8.10.3.38 percept_consp_number_seen()

```
elemental subroutine the_neurobio::percept_consp_number_seen (
            class(percept_conspecifics), intent(inout) this,
            integer, intent(in) number_set )
```

Set the total number of conspecifics perceived (seen) in the conspecific perception object. But do **not** reallocate the conspecific perception components so far.

Parameters

in	*number_set*	number_set Set the number of conspecifics in the perception object.

Definition at line 2283 of file m_neuro.f90.

8.10.3.39 percept_consp_make_fill_arrays()

```
pure subroutine the_neurobio::percept_consp_make_fill_arrays (
            class(percept_conspecifics), intent(inout) this,
            type(conspec_percept_comp), dimension(:), intent(in) consps )
```

Make the conspecifics perception object, fill it with the actual arrays.

Note

Note that the size and allocation is set by the `init` method.

Parameters

in	*consps*	consps an array of conspecific perception components that form the perception object.

PROCNAME is the procedure name for logging and debugging (with MODNAME).

8.10.3.39.1 Implementation details Fill the dynamic conspecific perception object with the data from the input array.
Definition at line 2297 of file m_neuro.f90.

8.10.3.40 percept_consp_get_count_seen()

```
elemental integer function the_neurobio::percept_consp_get_count_seen (
            class(percept_conspecifics), intent(in) this )
```

Get the number (count) of conspecifics seen. Trivial.
Definition at line 2316 of file m_neuro.f90.

8.10.3.41 percept_consp_destroy_deallocate()

```
elemental subroutine the_neurobio::percept_consp_destroy_deallocate (
            class(percept_conspecifics), intent(inout) this )
```

Deallocate and delete a conspecific perception object.
Definition at line 2326 of file m_neuro.f90.

8.10.3.42 consp_perception_get_visrange_objects()

```
subroutine the_neurobio::consp_perception_get_visrange_objects (
            class(perception), intent(inout) this,
```

```
class(condition), dimension(:), intent(in) consp_agents,
integer, intent(in), optional time_step_model )
```

Get available conspecific perception objects within the visual range of the agent, which the agent can perceive and therefore respond to.

Note

Note that there are three similar procedures that detect spatial objects within the visual range of the agent:

- the_neurobio::perception::see_food – perception of food items;
- the_neurobio::perception::see_consp – perception of conspecifics:
- the_neurobio::perception::see_pred – perception of predators.

All these procedures were actually implemented using the first (the_neurobio::perception::see_food) as a template. All three implement partial indexing of the nearest spatial objects to accelerate computation of large arrays of spatial objects.

Parameters

in	*consp_agents*	consp_agents An array of spatial objects where we are looking for the nearest available perception objects (array).
in	*time_step_model*	time_step_model The current time step of the model.

8.10.3.42.1 Notable variables and parameters

- **consp_sizes** - local array for the body lengths of the agents.

consp_masses - local array for the body masses of the agents.

- **consp_alive** - local array of logical indicators if the agents are alive (TRUE).
- **consp_sex_is_male** - local array of the sex of the conspecifics.
- **MIN_DIST_SELF** - exclude self from the neighbours. Because we cannot (easily) use recursive reference to the indvidal agent class from this lower-order perception class, we have to pass some parameters of the *this* agent as dummy parameters to the subroutine. E.g. individual ID, if ID is incorrect or not passed, the only way to distinguish self from other agents in the neighbourhood is by different location, i.e. the distance should be non-zero. This parameter sets the tolerance limit for the difference in the distance for considering the neighbour not-self. Possibly can be equal to the parameter `commondata::zero`, or we can allow a higher discrepancy (this also might correct some errors).
- **agents_near** - temporary array of nearby other conspecifics available to this agent.
- **dist_neighbours** - temporary array of the distances to the neighbouring food items.
- **dist_index** - temporary partial index vector for the distances to the neighbouring conspecifics.
- **irradiance_agent_depth** - local variable defining the irradiance (illumination) at the current depth of the agent. Needed to calculate the agent's visual range.
- **sobject_area** - local variable defining the conspecific area. Needed to calculate the agent's visual range.
- **sobject_visual_range** - local variable defining the visual range of the agent for detecting each of the conspecifics (with known areas) at the agent's current depth.
- **sobjects_percept_in_visrange** - local sorted array of conspecific perception components that are within the visual range of the agent for output. The array should normally have the size of `commondata::consp_select_items_index_partial` elements, but only the first `sobjects_n_visrange` elements of it are actually within the visual range.
- **sobjects_dist_sorted** - temporary local sorted array of distances between the agent and each of the nearest neighbouring conspecifics, sorted for output.

- **consp_array_size** - the size of the input arrays of object properties, local. Initially set from the size of the objects (class) array, but `consp_sizes` and `consp_alive` must have identical sizes.
- **sobjects_n_visrange** - local number of elements of `sobjects_percept_in_visrange` for output that are within he visual range of the agent.
- **self_idx** - local index of itself, needed to exclude self from debug messages and logs.

8.10.3.42.2 Implementation details

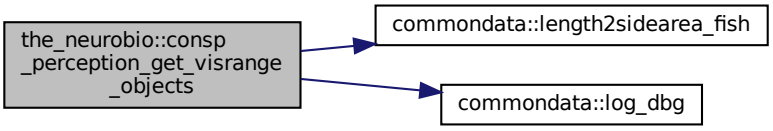
8.10.3.42.2.1 Checks and preparations Initialise index and rank values. Uninitialised index arrays may result in invalid memory reference in `ARRAY_INDEX` (it is not safe by design).
Also zero-init self index.
Check optional time step parameter. If unset, use global `commondata::global_time_step_model_current`.
Set the size for all the internal arrays, that is equal to the `consp_agents` objects array size.
Copy conspecifics array from the input `consp_agents` class into `agents_near`.
Get local arrays for the conspecific sizes, alive status and sex using elemental array-based accessor functions.

8.10.3.42.2.2 Step 1 First, we get, up to the maximum order (fast *partial indexing*) of `commondata::consp_select_items_` neighbouring conspecifics that are in proximity of this agent. here we get **partial index** vector for the input array of objects: `dist_index`. The calculation backend is the_environment::spatial::neighbours().

8.10.3.42.2.3 Step 2 Second, we select only those items within this set, which are within the visual range of the agent under the current conditions. To do this we, first, calculate the ambient illumination/irradiance level at the depth of the agent. Done using the the_environment::spatial::illumination() procedure.
Compute the array of "prey areas" for each conspecific.
Compute the vector of the visual ranges for detecting each of the conspecifics by the agent.

8.10.3.42.2.4 Step 3 Now we can get the pre-output array `sobjects_percept_in_visrange` that contains the conspecifics available within the visual range of the agent. Also, we count their number for the output parameter `sobjects_n_visrange`.
Include only agents non identical to oneself. TODO: Use individual ID, but have to pass as an additional dummy parameter.
Include only alive agents.
Index of itself, will exclude from min. values.
Here we also log warning if no conspecifics found, when debugging. If the self index self_idx is non-zero, will output next from self value, usually 2.

8.10.3.42.2.5 Step 4 Finally, we can now create the output conspecific perception object, including only conspecifics that are within the current visual range of the agent. Init (create and allocate) the conspecific perception object, at first empty, using the food perception object (at first empty) using the the_neurobio::percept_conspecifics::init().
Fill the output perception object with the values obtained at the step 3. This is done using the the_neurobio::percept_conspecifics::make backend procedure.
Definition at line 2347 of file m_neuro.f90.
Here is the call graph for this function:



8.10.3.43 consp_perception_is_seeing_conspecifics()

```
elemental logical function the_neurobio::consp_perception_is_seeing_conspecifics (
            class(perception), intent(in) this )
```

Check if the agent sees any conspecifics within the visual range.

Warning

Should be called after the `see_consp` method as it is only an accessor get-function.

Note

There is little sense to implement this accessor procedure in the specific perception (up-level) class as every perception is not a derivative of a common class, perceptions independent, so we'll have to also implement agent-bound perception methods anyway.

Definition at line 2614 of file m_neuro.f90.

8.10.3.44 spatialobj_percept_comp_create()

```
elemental subroutine the_neurobio::spatialobj_percept_comp_create (
            class(spatialobj_percept_comp), intent(inout) this )
```

Create a single arbitrary spatial object perception component at an undefined position with default properties.

8.10.3.44.1 Implementation notes Just set an undefined location of the object using standard interface function `the_environment::spatial::missing()`.
Set cid to commondata::unknown.

Warning

random id on create is now disabled to allow elemental function, because random are never pure. So care to set cid's elsewhere.

Then we set the object size to commondata::missing.
Init distance towards the object, initially also commondata::missing.
Definition at line 2630 of file m_neuro.f90.

8.10.3.45 spatialobj_percept_make()

```
subroutine the_neurobio::spatialobj_percept_make (
            class(spatialobj_percept_comp), intent(inout) this,
            type(spatial), intent(in) location,
            real(srp), intent(in), optional size,
            real(srp), intent(in), optional dist,
            integer, intent(in), optional cid )
```

Make a single arbitrary **spatial** object perception component.

Parameters

in	*location*	Location of the spatial object perception component, as a `the_environment::spatial` type container.
in	*size*	size This is the optional object size.
in	*dist*	dist The distance towards this object.
in	*cid*	iid Optional cid for the object, e.g. number within an array.

8.10.3.45.1 Implementation notes Set the location of the object using standard interface function `the_↩ envirnoment::spatial::position()`.
If individual id is provided, set it. If not, set random.
Set the object perception component size. Only nonzero size is accepted.
Also set the distance towards the conspecific if provided. If not provided, ignore.
Definition at line 2653 of file m_neuro.f90.

8.10.3.46 spatialobj_percept_get_size()

```
elemental real(srp) function the_neurobio::spatialobj_percept_get_size (
            class(spatialobj_percept_comp), intent(in) this )
```

Get an arbitrary spatial object perception component size.
Definition at line 2696 of file m_neuro.f90.

8.10.3.47 spatialobj_percept_get_dist()

```
elemental real(srp) function the_neurobio::spatialobj_percept_get_dist (
            class(spatialobj_percept_comp), intent(in) this )
```

Get the distance to an arbitrary spatial object perception component.
Definition at line 2707 of file m_neuro.f90.

8.10.3.48 spatialobj_percept_visibility_visual_range()

```
real(srp) function the_neurobio::spatialobj_percept_visibility_visual_range (
            class(spatialobj_percept_comp), intent(in) this,
            real(srp), intent(in), optional object_area,
            real(srp), intent(in), optional contrast,
            integer, intent(in), optional time_step_model )
```

Calculate the visibility range of this spatial object. Wrapper to the `visual_range` function. This function calculates the distance from which this object can be seen by a visual object (e.g. predator or prey).

Warning

The `visual_range` procedures use meter for units, this auto-converts to cm.

Cannot implement a generic function accepting also vectors of this objects as only elemental object-bound array functions are allowed by the standard. This function cannot be elemental, so passed-object dummy argument must always be scalar.

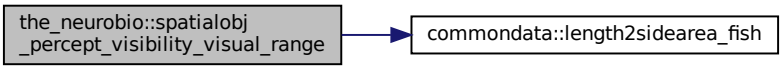
Parameters

in	*object_area*	object_area is optional area of the spatial object. This parameter can be necessary because the area can be calculated differently for different types of objects, e.g. fish using commondata::length2sidearea_fish() or food items using commondata::carea(). The default object area if the parameter is absent, uses the fish backend.
in	*contrast*	contrast is an optional inherent visual contrast of the spatial object. The default contrast of all objects is defined by the commondata::preycontrast_default parameter.
in	*time_step_model*	optional time step of the model, if absent gets the current time step as defined by the value of `commondata::global_time_step_model_current`.

Returns

The maximum distance from which this object can be seen.

8.10.3.48.1 Implementation details Check optional `contrast` parameter. If unset, use global `commondata::preycontras`
Check optional time step parameter. If unset, use global `commondata::global_time_step_model_current`.
Second, check if the object area (`object_area`) parameter is provided. If not, calculate area from the `sobj↩_size` component assuming it is a fish. This is logical because the_neurobio::spatialobj_percept_comp class is mainly used in the perception of conspecifics and predators.
Calculate ambient illumination / irradiance at the depth of this object at the given time step.
Return visual range to see this spatial object: its visibility range.
Definition at line 2727 of file m_neuro.f90.
Here is the call graph for this function:



8.10.3.49 percept_predator_create_init()

```
elemental subroutine the_neurobio::percept_predator_create_init (
            class(percept_predator), intent(inout) this,
            integer, intent(in) maximum_number_predators )
```

Create **predator** perception object, it is an array of spatial perception components.

Parameters

in	*maximum_number_predators*	maximum_number_predators The maximum number of predators in the perception object. Normally equal to the partial predator selection indexing order `PREDATOR_SELECT_ITEMS_INDEX_PARTIAL`.

8.10.3.49.1 Implementation notes Allocate the array of the predator's spatial perception components.
Allocate the array of the predator attack rates.
And create all perception components (create is `elemental`).
Fill the predator's attack rates arrays with commondata::missing values.
Set the initial number of predators within the visual range to the maximum number provided.
Definition at line 2799 of file m_neuro.f90.

8.10.3.50 percept_predator_number_seen()

```
elemental subroutine the_neurobio::percept_predator_number_seen (
            class(percept_predator), intent(inout) this,
            integer, intent(in) number_set )
```

Set the total number of **predators** perceived (seen) in the predator perception object. But do not reallocate the predator perception components so far.

Parameters

in	*number_set*	number_set Set the number of predators in the perception object.

Definition at line 2834 of file m_neuro.f90.

8.10.3.51 percept_predator_make_fill_arrays()

```
pure subroutine the_neurobio::percept_predator_make_fill_arrays (
            class(percept_predator), intent(inout) this,
            type(spatialobj_percept_comp), dimension(:), intent(in) preds,
            real(srp), dimension(:), intent(in), optional attack_rate )
```

Make the **predator** perception object, fill it with the actual arrays.

Note

Note that the size and allocation is set by the the_neurobio::percept_predator::init() method.

Parameters

in	*preds*	preds an array of predator (spatial, the_neurobio::spatialobj_percept_comp) perception components that form the perception object.

8.10.3.51.1 Implementation notes Fill the dynamic conspecific perception object with the data from the input array.
The arrays for the body sizes and attack rates of the predators are set only if these arrays are present in the dummy arguments to this procedure. If they are not provided, default values are used as defined by commondata::predator_attack_rate_default parameter.
Definition at line 2849 of file m_neuro.f90.

8.10.3.52 percept_predator_set_attack_rate_vector()

```
pure subroutine the_neurobio::percept_predator_set_attack_rate_vector (
            class(percept_predator), intent(inout) this,
            real(srp), dimension(:), intent(in) attack_rate )
```

Set an array of the attack rates for the predator perception object.
Definition at line 2881 of file m_neuro.f90.

8.10.3.53 percept_predator_set_attack_rate_scalar()

```
pure subroutine the_neurobio::percept_predator_set_attack_rate_scalar (
            class(percept_predator), intent(inout) this,
            real(srp), intent(in) attack_rate )
```

Set an array of the attack rates for the predator perception object.
Definition at line 2892 of file m_neuro.f90.

8.10.3.54 percept_predator_get_count_seen()

```
elemental integer function the_neurobio::percept_predator_get_count_seen (
            class(percept_predator), intent(in) this )
```

Get the number (count) of predators seen. Trivial.
Definition at line 2903 of file m_neuro.f90.

8.10.3.55 percept_predator_destroy_deallocate()

```
elemental subroutine the_neurobio::percept_predator_destroy_deallocate (
            class(percept_predator), intent(inout) this )
```

Deallocate and delete a **predator** perception object.
Definition at line 2913 of file m_neuro.f90.

8.10.3.56 predator_perception_get_visrange_objects()

```
subroutine the_neurobio::predator_perception_get_visrange_objects (
            class(perception), intent(inout) this,
            class(predator), dimension(:), intent(in) spatl_agents,
            integer, intent(in), optional time_step_model )
```

Get available predators perception objects within the visual range of the agent, which the agent can perceive and therefore respond to.

Note

Note that there are three similar procedures that detect spatial objects within the visual range of the agent:

- the_neurobio::perception::see_food – perception of food items;
- the_neurobio::perception::see_consp – perception of conspecifics:
- the_neurobio::perception::see_pred – perception of predators.

All these procedures were actually implemented using the first (the_neurobio::perception::see_food) as a template. All three implement partial indexing of the nearest spatial objects to accelerate computation of large arrays of spatial objects.

Note

This procedure also used the conspecific perception as a template, but removed extra tests for "self".

Parameters

in	*spatl_agents*	spatl_agents An array of spatial objects where we are looking for the nearest available perception objects (array).
in	*time_step_model*	time_step_model The current time step of the model.

8.10.3.56.1 Notable variables and parameters

- **MIN_DIST_SELF** - exclude self from the neighbours. Because we cannot (easily) use recursive reference to the individual agent class from this lower-order perception class, we have to pass some parameters of the *this* agent as dummy parameters to the subroutine. E.g. individual ID, if ID is incorrect or not passed, the only way to distinguish self from other agents in the neighbourhood is by different location, i.e. the distance should be non-zero. This parameter sets the tolerance limit for the difference in the distance for considering the neighbour not-self. Possibly can be equal to the parameter `commondata::zero`, or one can allow a higher discrepancy (this also might correct some errors).

agents_near - temporary array of predators in proximity to this agent.

- **dist_neighbours** - temporary array of the distances to the neighbouring predators.
- **dist_index** - temporary partial index vector for the distances to the neighbouring predators.
- **irradiance_agent_depth** - local variable defining the irradiance (illumination) at the current depth of the agent. Needed to calculate the agent's visual range.
- **spatl_sizes** -local copy of the body lengths of the predators.
- **sobject_area** - local variable defining the conspecific area. Needed to calculate the agent's visual range.

- **sobject_visual_range** - local variable defining the visual range of the agent for detecting each of the predators (with known areas) at the agent's current depth.
- **sobjects_percept_in_visrange** - local sorted array of the perception components that are within the visual range of the agent for output. The array should normally have the size of commondata::pred_select_items_index_p elements, but only the first sobjects_n_visrange elements of it are actually within the visual range.
- **pred_attack_rates_in_visrange** - local variable containing the attack rates of the predators that are within the visual range of the agent. The array should normally have the size of commondata::pred_select_items_index_p elements, but only the first sobjects_n_visrange elements of it are actually within the visual range.
- **index_max_size** - the size of the input arrays of object properties, local. Initially set from the size of the objects (class) array, but spatl_sizes must have identical size.
- **sobjects_n_visrange** - local number of elements of food_items_percept_in_visrange for output that are within he visual range of the agent.

8.10.3.56.2 Implementation details

8.10.3.56.2.1 Checks and preparations Initialise index and rank values. Uninitialised index arrays may result in invalid memory reference in ARRAY_INDEX (it is not safe by design).
Check optional time step parameter. If unset, use global commondata::global_time_step_model_current.
This is the maximum size of the index. If the number of spatial objects is huge, we use partial indexing of the neighbours array. Then it is equal to the partial indexing commondata::pred_select_items_index_partial.
However, if the number of potential neighbouring objects is smaller than the partial index size, we use full indexing. In the later case index_max_size is equal to the actual size of the neighbouring objects array.

Note

Distinguishing between the partial indexing parameter and the max size of the index makes sense only in cases where small number of neighbours can be expected. We normally have large populations of agents and huge number of food items, well exceeding the partial indexing parameter commondata::pred_select_items_index_partial. However, the number of predators can be smaller than this.

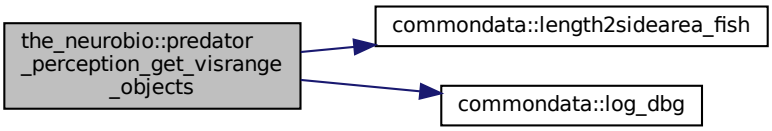
Copy predators array from the input spatl_agents class into agents_near.
Get an array of the body sizes of the predators using array-based elemental function (This now works ok in Intel Fortran 17).
Get an array of the attack rates of the predators using array-based elemental function.

8.10.3.56.2.2 Step 1 First, we get, up to the maximum order (fast *partial indexing*) of commondata::pred_select_items_i neighbouring predators that are in proximity of this agent. here we get partial index vector for the input array of objects: dist_index. **Partial indexing** is the most typical case as we have normally quite large number of agents within the population and food items within the habitat. However, this might be important for **predators**. Predators can be quite infrequent within the habitat, their number can be smaller than the maximum indexing parameter commondata::pred_select_items_index_partial. Hence the check is much more important here than in similar procedures for food items and conspecifics. The neighbours are computed using the the_environment::spatial::neighbours() procedure.
Here we check if the number of neighbouring agents is smaller than the partial indexing parameter commondata::pred_select_items_index_partial and if yes, do **full indexing**.
However, if the number of potential neighbouring objects is big, do **partial indexing**.

8.10.3.56.2.3 Step 2 Second, we select only those items within this set, which are within the visual range of the agent under the current conditions. To do this we, first, calculate the ambient illumination/irradiance level at the depth of the agent. Done using the the_environment::spatial::illumination() procedure.
Compute the array of "prey areas" for each conspecific. So far prey_contrast is not set, use default value commondata::individual_visual_contrast_default.
Compute the vector of the visual ranges for detecting each of the predators by the agent.

8.10.3.56.2.4 Step 3 Now we can get the pre-output array `sobjects_percept_in_visrange` that contains the predators available within the visual range of the agent. Also, we count their number for the output parameter `sobjects_n_visrange`.
The inherent attack rates of each of the predators within the visual range is also collected here into the `pred_attack_rates_in_visrange` array.
Here we also log warning if no objects found, when debugging. (see commondata::is_debug).

8.10.3.56.2.5 Step 4 Finally, we can now create the output conspecific perception object, including only predators that are within the current visual range of the agent. Init (create and allocate) the predator perception object, at first empty, with the_neurobio::percept_predator::init().
Fill the output perception object with the values obtained at the step 3 using the the_neurobio::percept_predator::make() method.
Note that if the commondata::agent_can_assess_predator_attack_rate parameter is set to TRUE, the agents can access and assess the inherent attack rate of the predator. This can be for example possible if the agent can assess the hunger level of the predator. The attack rate is then set from the array of the inherent attack rates of the predators `pred_attack_rates_in_visrange`. May need to add a predator perception error to this value, but it is not implemented yet.
Note that if the commondata::agent_can_assess_predator_attack_rate parameter is set to FALSE, the agents cannot access the inherent attack rate of the predator. The attack rate is then fixed from the default parameter commondata::predator_attack_rate_default value.
Definition at line 2937 of file m_neuro.f90.
Here is the call graph for this function:



8.10.3.57 predator_perception_is_seeing_predators()

```
elemental logical function the_neurobio::predator_perception_is_seeing_predators (
            class(perception), intent(in) this )
```

Check if the agent sees any predators within the visual range.

Warning

Should be called after the `the_neurobio::perception::see_food()` method as it is just an accessor function.

Definition at line 3226 of file m_neuro.f90.

8.10.3.58 percept_light_create_init()

```
elemental subroutine the_neurobio::percept_light_create_init (
            class(percept_light), intent(inout) this )
```

Make en empty light perception component. Really necessary only when perception objects are all allocatable.
Definition at line 3243 of file m_neuro.f90.

8.10.3.59 percept_light_get_current()

```
elemental real(srp) function the_neurobio::percept_light_get_current (
            class(percept_light), intent(in) this )
```

Get the current perception of the illumination.
Definition at line 3252 of file m_neuro.f90.

8.10.3.60 percept_light_set_current()

```
subroutine the_neurobio::percept_light_set_current (
            class(percept_light), intent(inout) this,
            integer, intent(in) timestep,
            real(srp), intent(in) depth )
```

Set the current **light** level into the perception component.
Here we, calculate the ambient illumination/irradiance level at the current depth of the agent.

Note

`is_stochastic` logical parameter is TRUE in `light_surface`, that sets a stochastic illumination level at the surface and therefore also at the agent's current depth.

Definition at line 3262 of file m_neuro.f90.

8.10.3.61 percept_light_destroy_deallocate()

```
elemental subroutine the_neurobio::percept_light_destroy_deallocate (
            class(percept_light), intent(inout) this )
```

Destroy / deallocate **light** perception component. Really necessary only when perception objects are all allocatable.
Definition at line 3283 of file m_neuro.f90.

8.10.3.62 light_perception_get_object()

```
subroutine the_neurobio::light_perception_get_object (
            class(perception), intent(inout) this,
            integer, intent(in), optional time_step_model )
```

Get **light** perception objects into the individual PERCEPTION object layer.

Parameters

in	*time_step_model*	time_step_model The current time step of the model.

Local copy of the time step of the model.
Check optional time step parameter. If unset, use global `commondata::global_time_step_model_current`.
Allocate and init the perception object (needed only when it is allocatable)
Definition at line 3293 of file m_neuro.f90.

8.10.3.63 percept_depth_create_init()

```
elemental subroutine the_neurobio::percept_depth_create_init (
            class(percept_depth), intent(inout) this )
```

Make en empty depth perception component. Really necessary only when perception objects are all allocatable.
Definition at line 3325 of file m_neuro.f90.

8.10.3.64 percept_depth_get_current()

```
elemental real(srp) function the_neurobio::percept_depth_get_current (
            class(percept_depth), intent(in) this )
```

Get the current perception of the **depth**.
Definition at line 3334 of file m_neuro.f90.

8.10.3.65 percept_depth_set_current()

```
subroutine the_neurobio::percept_depth_set_current (
            class(percept_depth), intent(inout) this,
            real(srp), intent(in) cdepth )
```

Set the current **depth** level into the perception component.
Definition at line 3344 of file m_neuro.f90.

8.10.3.66 percept_depth_destroy_deallocate()

```
elemental subroutine the_neurobio::percept_depth_destroy_deallocate (
            class(percept_depth), intent(inout) this )
```

Destroy / deallocate **depth** perception component. Really necessary only when perception objects are all allocatable.
Definition at line 3355 of file m_neuro.f90.

8.10.3.67 percept_reprfac_create_init()

```
elemental subroutine the_neurobio::percept_reprfac_create_init (
            class(percept_reprfact), intent(inout) this )
```

Make en empty reproductive factor perception component. Really necessary only when perception objects are all allocatable.
Definition at line 3369 of file m_neuro.f90.

8.10.3.68 percept_reprfac_get_current()

```
elemental real(srp) function the_neurobio::percept_reprfac_get_current (
            class(percept_reprfact), intent(in) this )
```

Get the current perception of the **reproductive factor**.
Definition at line 3378 of file m_neuro.f90.

8.10.3.69 percept_reprfac_set_current()

```
subroutine the_neurobio::percept_reprfac_set_current (
            class(percept_reprfact), intent(inout) this,
            real(srp), intent(in) reprfac )
```

Set the current **reproductive factor** level into perception component.
Definition at line 3388 of file m_neuro.f90.

8.10.3.70 percept_reprfac_destroy_deallocate()

```
elemental subroutine the_neurobio::percept_reprfac_destroy_deallocate (
            class(percept_reprfact), intent(inout) this )
```

Destroy / deallocate **reproductive factor** perception component. Really necessary only when perception objects are all allocatable.
Definition at line 3399 of file m_neuro.f90.

8.10.3.71 depth_perception_get_object()

```
subroutine the_neurobio::depth_perception_get_object (
            class(perception), intent(inout) this )
```

Get **depth** perception objects into the **individual** PERCEPTION object layer.
Allocate and init the perception object (needed only when it is allocatable)
Definition at line 3413 of file m_neuro.f90.

8.10.3.72 stomach_perception_get_object()

```
subroutine the_neurobio::stomach_perception_get_object (
            class(perception), intent(inout) this )
```

Get the **stomach capacity** perception objects into the **individual** PERCEPTION object layer.
Allocate and init the perception object (needed only when it is allocatable)
Calculate the available stomach capacity, i.e. maximum stomach mass minus current stomach content, and set the value into the stomach perception object. Body mass and maxstomcap are from the CONDITION layer.
Definition at line 3427 of file m_neuro.f90.

8.10.3.73 bodymass_perception_get_object()

```
subroutine the_neurobio::bodymass_perception_get_object (
            class(perception), intent(inout) this )
```

Get the **body mass** perception objects into the **individual** PERCEPTION object layer.
Allocate and init the perception object (needed only when it is allocatable)
Get the body mass from the individual, put it to the perception object.
Definition at line 3446 of file m_neuro.f90.

8.10.3.74 energy_perception_get_object()

```
subroutine the_neurobio::energy_perception_get_object (
            class(perception), intent(inout) this )
```

Get the **energy reserves** perception objects into the **individual** PERCEPTION object layer.
Allocate and init the perception object (needed only when it is allocatable)
Get the energy reserves from the individual, put it to the perception object.
Definition at line 3461 of file m_neuro.f90.

8.10.3.75 age_perception_get_object()

```
subroutine the_neurobio::age_perception_get_object (
            class(perception), intent(inout) this )
```

Get the **age** perception objects into the **individual** PERCEPTION object layer.
Allocate and init the perception object (needed only when it is allocatable).
Get the age from the individual, put it to the perception object.
Definition at line 3477 of file m_neuro.f90.

8.10.3.76 repfac_perception_get_object()

```
subroutine the_neurobio::repfac_perception_get_object (
            class(perception), intent(inout) this )
```

Get the **reproductive factor** perception objects into the **individual** PERCEPTION object layer.
Allocate and init the perception object (needed only when it is allocatable).
Get the the_hormones::hormones::reproductive_factor() from the individual, put it to the perception object.
Definition at line 3493 of file m_neuro.f90.

8.10.3.77 percept_memory_add_to_stack()

```
elemental subroutine the_neurobio::percept_memory_add_to_stack (
            class(memory_perceptual), intent(inout) this,
            real(srp), intent(in) light,
            real(srp), intent(in) depth,
            real(srp), intent(in) food,
            real(srp), intent(in) foodsize,
            real(srp), intent(in) fooddist,
            integer, intent(in) consp,
            integer, intent(in) pred,
            real(srp), intent(in) stom,
            real(srp), intent(in) bdmass,
            real(srp), intent(in) energ,
            real(srp), intent(in) reprfac )
```

Add perception components into the memory stack.

Parameters

in	*light*	The parameters of the subroutine are the actual values that are added to the perceptual memory stack arrays.

Each of the memory stack components corresponds to the respective dummy parameter. So arrays are updated at each step.
Definition at line 3513 of file m_neuro.f90.

8.10.3.78 percept_memory_cleanup_stack()

```
elemental subroutine the_neurobio::percept_memory_cleanup_stack (
            class(memory_perceptual), intent(inout) this )
```

Cleanup and destroy the perceptual memory stack.
cleanup procedure uses whole array assignment.
Definition at line 3550 of file m_neuro.f90.

8.10.3.79 percept_memory_food_get_total()

```
elemental integer function the_neurobio::percept_memory_food_get_total (
            class(memory_perceptual), intent(in) this )
```

Get the total number of food items within the whole perceptual memory stack.

Returns

Total count of predators in the memory stack.

Calculate the overall sum excluding missing values (masked).
Definition at line 3572 of file m_neuro.f90.

8.10.3.80 percept_memory_food_get_mean_n()

```
elemental real(srp) function the_neurobio::percept_memory_food_get_mean_n (
            class(memory_perceptual), intent(in) this,
            integer, intent(in), optional last )
```

Get the **average number** of food items per single time step within the whole perceptual memory stack.

Note

There are several similar procedures with very similar implementation:

- the_neurobio::percept_memory_food_get_mean_n() - get mean **number** of food items from the memory;
- the_neurobio::percept_memory_food_get_mean_size() - get mean **size** of food items from the memory.
- the_neurobio::percept_memory_predators_get_mean() - get average number of predators from the memory.

Parameters

in	*last*	last Limit to only this number of latest components in history.

Returns

Mean count of food items in the memory stack.

Local copy of optional last

8.10.3.80.1 Implementation notes Check if we are given the parameter requesting the latest history size. if parameter `last` absent or bigger than the array size, get whole stack array.
Calculate the average excluding missing values (masked) using commondata::average().
Definition at line 3595 of file m_neuro.f90.

8.10.3.81 percept_memory_food_mean_n_split()

```
elemental subroutine the_neurobio::percept_memory_food_mean_n_split (
            class(memory_perceptual), intent(in) this,
            integer, intent(in), optional window,
            integer, intent(in), optional split_val,
            real(srp), intent(out) older,
            real(srp), intent(out) newer )
```

Get the **average number** of food items per single time step within the perceptual memory stack, split to the first (older) and second (newer) parts. The whole memory stack ('sample') is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it.

Note

There are several similar procedures with very similar implementation:

- the_neurobio::percept_memory_food_mean_n_split() - get mean **number** of food items from the memory;
- the_neurobio::percept_memory_food_mean_size_split() - get mean **size** of food items from the memory;
- the_neurobio::percept_memory_predators_mean_split() - get average number of predators.

Parameters

in	*window*	window is the whole memory window which is analysed, if not present, the whole memory stack is used.
in	*split_val*	split_val is the split value for the separation of the older and newer averages. If not present, just splits the memory window evenly in two halves.
out	*older*	older is the output average number of the food items in the first (older) part of the memory window.
out	*newer*	newer is the output average number of the food items in the second (newer) part of the memory window.

8.10.3.81.1 Implementation details First, check optional parameters: the memory window `window` and the split value `split_val`. If either is not provided, defaults are used.
(Also, a check is made so that a window exceeding the history stack length is reduced accordingly to the whole memory size).

- whole size of the perceptual memory stack commondata::history_size_perception for the memory window
- half of the memory window for the `split_val`.

A sanity check is also done, if the split value happen to exceed the `window` parameter, it is reduced to the default 1/2 of the `window`.
Second, the `older` and the `newer` output average values are calculated. Here is the illustration of the calculation:

```
Such 'window' and 'split_val'
values...

          |<----- window ----->|
+---------+--------------------+
+         |         :|:         +
+---------+--------------------+
                     ^ split_val


... result in these means:

+---------+---------------------+
+         | mean for | mean for +
+         | 'older'  | 'newer'  +
+---------+---------------------+
```

Definition at line 3648 of file m_neuro.f90.

8.10.3.82 percept_memory_food_get_mean_size()

```
elemental real(srp) function the_neurobio::percept_memory_food_get_mean_size (
             class(memory_perceptual), intent(in) this,
             integer, intent(in), optional last )
```

Get the **average size** of food item per single time step within the whole perceptual memory stack.

Note

There are several similar procedures with very similar implementation:

- the_neurobio::percept_memory_food_get_mean_n() - get mean **number** of food items from the memory;
- the_neurobio::percept_memory_food_get_mean_size() - get mean **size** of food items from the memory.
- the_neurobio::percept_memory_predators_get_mean() - get average number of predators from the memory.

Parameters

in	*last*	last Limit to only this number of latest components in history.

Returns

Mean size of food items in the memory stack.

Local copy of optional last

8.10.3.82.1 Implementation notes Check if we are given the parameter requesting the latest history size. if parameter `last` absent or bigger than the array size, get whole stack array.
Calculate the average excluding missing values (masked) using commondata::average().

Definition at line 3739 of file m_neuro.f90.

8.10.3.83 percept_memory_food_mean_size_split()

```
elemental subroutine the_neurobio::percept_memory_food_mean_size_split (
            class(memory_perceptual), intent(in) this,
            integer, intent(in), optional window,
            integer, intent(in), optional split_val,
            real(srp), intent(out) older,
            real(srp), intent(out) newer )
```

Get the **average size** of food items per single time step within the perceptual memory stack, split to the first (older) and second(newer) parts. The whole memory stack 'sample' is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it.

Note

There are several similar procedures with very similar implementation:

- the_neurobio::percept_memory_food_mean_n_split() - get mean **number** of food items from the memory;
- the_neurobio::percept_memory_food_mean_size_split() - get mean **size** of food items from the memory;
- the_neurobio::percept_memory_predators_mean_split() - get average number of predators.

Parameters

`in`	*window*	window is the whole memory window which is analysed, if not present, the whole memory stack is used.
`in`	*split_val*	split_val is the split value for the separation of the older and newer averages. If not present, just splits the memory window evenly in two halves.
`out`	*older*	older is the output average sizes of the food items in the first (older) part of the memory window.
`out`	*newer*	newer is the output average sizes of the food items in the second (newer) part of the memory window.

8.10.3.83.1 Implementation details First, check optional parameters: the memory window `window` and the split value `split_val`. If either is not provided, defaults are used.
(Also, a check is made so that a window exceeding the history stack length is reduced accordingly to the whole memory size).

- whole size of the perceptual memory stack commondata::history_size_perception for the memory window
- half of the memory window for the `split_val`.

A sanity check is also done, if the split value happen to exceed the `window` parameter, it is reduced to the default 1/2 of the `window`.
Second, the `older` and the `newer` output average values are calculated. Here is the illustration of the calculation:

```
Such 'window' and 'split_val'
values...

          |<----- window ----->|
+---------+--------------------+
+         |         :|:         +
+---------+--------------------+
                     ^ split_val


... result in these means:
```

```
+--------+---------------------+
+        | mean for | mean for +
+        | 'older'  | 'newer'  +
+--------+---------------------+
```

Definition at line 3792 of file m_neuro.f90.

8.10.3.84 percept_memory_food_get_mean_dist()

```
elemental real(srp) function the_neurobio::percept_memory_food_get_mean_dist (
            class(memory_perceptual), intent(in) this,
            integer, intent(in), optional last,
            logical, intent(in), optional undef_ret_null )
```

Get the **average distance** to food item per single time step within the whole perceptual memory stack.

Note

There are several similar procedures with very similar implementation:

- the_neurobio::percept_memory_food_get_mean_n() - get mean **number** of food items from the memory;
- the_neurobio::percept_memory_food_get_mean_size() - get mean **size** of food items from the memory.
- the_neurobio::percept_memory_predators_get_mean() - get average number of predators from the memory.

Parameters

in	*last*	last Limit to only this number of latest components in history.
in	*undef_ret_null*	undef_ret_null Optional flag if undefined value with sample size should return zero mean value; if absent is set to TRUE and zero mean is returned. Note that this behaviour is the opposite of the standard commondata::average(). It is because this function is mainly for perception memory, where zero value is appropriate in absence of any food items.

Returns

Mean distance to food items in the memory stack.

Local copies of optionals

8.10.3.84.1 Implementation notes Check if we are given the parameter requesting the latest history size. if parameter `last` absent or bigger than the array size, get whole stack array.
Calculate the average excluding missing values (masked) using commondata::average().
Definition at line 3883 of file m_neuro.f90.

8.10.3.85 percept_memory_food_mean_dist_split()

```
elemental subroutine the_neurobio::percept_memory_food_mean_dist_split (
            class(memory_perceptual), intent(in) this,
            integer, intent(in), optional window,
            integer, intent(in), optional split_val,
            real(srp), intent(out) older,
            real(srp), intent(out) newer )
```

Get the **average distance** to food items per single time step within the perceptual memory stack, split to the first (older) and second(newer) parts. The whole memory stack 'sample' is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it.

Note

There are several similar procedures with very similar implementation:

- the_neurobio::percept_memory_food_mean_n_split() - get mean **number** of food items from the memory;
- the_neurobio::percept_memory_food_mean_size_split() - get mean **size** of food items from the memory;
- the_neurobio::percept_memory_predators_mean_split() - get average number of predators.

Parameters

in	*window*	window is the whole memory window which is analysed, if not present, the whole memory stack is used.
in	*split_val*	split_val is the split value for the separation of the older and newer averages. If not present, just splits the memory window evenly in two halves.
out	*older*	older is the output average distance to the food items in the first (older) part of the memory window.
out	*newer*	newer is the output average distance to the food items in the second (newer) part of the memory window.

8.10.3.85.1 Implementation details First, check optional parameters: the memory window `window` and the split value `split_val`. If either is not provided, defaults are used.
(Also, a check is made so that a window exceeding the history stack length is reduced accordingly to the whole memory size).

- whole size of the perceptual memory stack commondata::history_size_perception for the memory window
- half of the memory window for the `split_val`.

A sanity check is also done, if the split value happen to exceed the `window` parameter, it is reduced to the default 1/2 of the `window`.
Second, the `older` and the `newer` output average values are calculated. Here is the illustration of the calculation:

```
Such 'window' and 'split_val'
values...

          |<----- window ----->|
+---------+--------------------+
+         |         :|:         +
+---------+--------------------+
                     ^ split_val


... result in these means:

+---------+----------------------+
+         | mean for | mean for +
+         | 'older'  | 'newer'  +
+---------+----------------------+
```

Definition at line 3948 of file m_neuro.f90.

8.10.3.86 percept_memory_consp_get_mean_n()

```
elemental real(srp) function the_neurobio::percept_memory_consp_get_mean_n (
             class(memory_perceptual), intent(in) this,
             integer, intent(in), optional last )
```

Get the **average number** of conspecifics per single time step within the whole perceptual memory stack.

Parameters

in	*last*	last Limit to only this number of latest components in history.

Returns

Mean count of conspecifics in the memory stack.

Local copy of optional last

8.10.3.86.1 Implementation notes Check if we are given the parameter requesting the latest history size. if parameter `last` absent or bigger than the array size, get whole stack array.
Calculate the average excluding missing values (masked) using commondata::average().
Definition at line 4030 of file m_neuro.f90.

8.10.3.87 percept_memory_predators_get_total()

```
elemental integer function the_neurobio::percept_memory_predators_get_total (
            class(memory_perceptual), intent(in) this )
```

Get the total number of predators within the whole perceptual memory stack.

Returns

Total count of predators in the memory stack.

Calculate the overall sum excluding missing values (masked).
Definition at line 4070 of file m_neuro.f90.

8.10.3.88 percept_memory_predators_get_mean()

```
elemental real(srp) function the_neurobio::percept_memory_predators_get_mean (
            class(memory_perceptual), intent(in) this,
            integer, intent(in), optional last )
```

Get the average number of predators per single time step within the whole perceptual memory stack.

Note

There are several similar procedures with very similar implementation:

- the_neurobio::percept_memory_food_get_mean_n() - get mean **number** of food items from the memory;
- the_neurobio::percept_memory_food_get_mean_size() - get mean **size** of food items from the memory.
- the_neurobio::percept_memory_predators_get_mean() - get average number of predators from the memory.

Parameters

in	*last*	last Limit to only this number of latest components in the history.

Returns

Mean count of predators in the memory stack.

Local copy of optional last
History stack size. We determine it from the size of the actual array rather than `HISTORY_SIZE_PERCEPTION` for further safety.
Check if we are given the parameter requesting the latest history size. if parameter `last` absent or bigger than the array size, get whole stack array.

Calculate the average excluding missing values (masked).
Definition at line 4093 of file m_neuro.f90.

8.10.3.89 percept_memory_predators_mean_split()

```
elemental subroutine the_neurobio::percept_memory_predators_mean_split (
            class(memory_perceptual), intent(in) this,
            integer, intent(in), optional window,
            integer, intent(in), optional split_val,
            real(srp), intent(out) older,
            real(srp), intent(out) newer )
```

Get the **average number** of predators per single time step within the perceptual memory stack, split to the first (older) and second(newer) parts. The whole memory stack ('sample') is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it.

Note

There are several similar procedures with very similar implementation:

- the_neurobio::percept_memory_food_mean_n_split() - get mean **number** of food items from the memory;
- the_neurobio::percept_memory_food_mean_size_split() - get mean **size** of food items from the memory;
- the_neurobio::percept_memory_predators_mean_split() - get average number of predators.

Parameters

`in`	*window*	window is the whole memory window which is analysed, if not present, the whole memory stack is used.
`in`	*split_val*	split_val is the split value for the separation of the older and newer averages. If not present, just splits the memory window evenly in two halves.
`out`	*older*	older is the output average number of predators in the first (older) part of the memory window.
`out`	*newer*	newer is the output average number of predators in the second (newer) part of the memory window.

8.10.3.89.1 Implementation details First, check optional parameters: the memory window `window` and the split value `split_val`. If either is not provided, defaults are used.
(Also, a check is made so that a window exceeding the history stack length is reduced accordingly to the whole memory size).

- whole size of the perceptual memory stack commondata::history_size_perception for the memory window
- half of the memory window for the `split_val`.

A sanity check is also done, if the split value happen to exceed the `window` parameter, it is reduced to the default 1/2 of the `window`.
Second, the `older` and the `newer` output average values are calculated. Here is the illustration of the calculation:

```
Such 'window' and 'split_val'
values...

          |<----- window ----->|
+---------+--------------------+
+         |         :|:         +
+---------+--------------------+
                    ^ split_val
```

```
... result in these means:

+--------+--------------------+
+        | mean for | mean for +
+        | 'older'  | 'newer'  +
+--------+--------------------+
```

Definition at line 4145 of file m_neuro.f90.

8.10.3.90 perception_objects_add_memory_stack()

```
elemental subroutine the_neurobio::perception_objects_add_memory_stack (
            class(perception), intent(inout) this )
```

Add the various perception objects to the memory stack object. This procedure is called **after** all the perceptual components (light, depth food, conspecifics, predators, etc.) are collected (using `set` object-bound subroutines) into the perception bundle, so all the values are known and ready to be used.
Now collect all perception variables into the whole memory stack object.
Definition at line 4230 of file m_neuro.f90.

8.10.3.91 perception_objects_get_all_environmental()

```
subroutine the_neurobio::perception_objects_get_all_environmental (
            class(perception), intent(inout) this )
```

A single umbrella subroutine to get all **environmental** perceptions: light, depth. This procedure invokes these calls:

- the_neurobio::perception::feel_light()
- the_neurobio::perception::feel_depth()

See also the_neurobio::perception::perceptions_inner().
Definition at line 4260 of file m_neuro.f90.

8.10.3.92 perception_objects_get_all_inner()

```
subroutine the_neurobio::perception_objects_get_all_inner (
            class(perception), intent(inout) this )
```

A single umbrella subroutine wrapper to get all **inner** perceptions: stomach, body mass, energy, age. Invokes all these procedures:

- the_neurobio::perception::feel_stomach()
- the_neurobio::perception::feel_bodymass()
- the_neurobio::perception::feel_energy()
- the_neurobio::perception::feel_age()
- the_neurobio::perception::feel_repfac()

See also the_neurobio::perception::perceptions_environ().
Splitting between the procedures for getting the inner and outer perceptions is for convenience only, this inner perceptions subroutine has no other parameters.

Warning

> It would **not** be easy to implement such a wrapper for the **outer** perceptions because the population and various environmental objects are not yet available at this object level.

Note

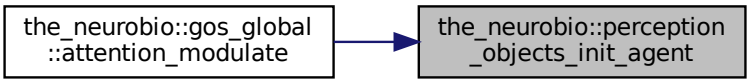
Templates for outer environmental perceptions: `call proto_parentsindividual(ind)see_food( & food_↩ resource_available = habitat_safefood, & time_step_model = 1)`

`call proto_parentsindividual(ind)see_consp( & consp_agents = proto_parentsindividual, & time_step_model = 1 )` `call proto_parentsindividual(ind)see_pred( & spatl_agents = predators, & time_step_model = 1 )call proto_↩ parentsindividual(ind)feel_light(timestep)call proto_parentsindividual(ind)feel_depth()`
Definition at line 4299 of file m_neuro.f90.

8.10.3.93 perception_objects_init_agent()

```
elemental subroutine, private the_neurobio::perception_objects_init_agent (
            class(perception), intent(inout) this )  [private]
```

Initialise all the perception objects for the current agent. Do not fill perception objects with the real data yet.
Init all perception objects within the agent.
Init and cleanup perceptual memory stack at start.
Definition at line 4313 of file m_neuro.f90.
Here is the caller graph for this function:



8.10.3.94 perception_objects_destroy()

```
elemental subroutine the_neurobio::perception_objects_destroy (
            class(perception), intent(inout) this,
            logical, intent(in), optional clean_memory )
```

Destroy and deallocate all perception objects.

Parameters

in	*clean_memory*	clean_memory Logical flag to cleanup perceptual memory stack.

Use the `destroy` method for all perception objects within the agent.
Init and cleanup perceptual memory stack if clean_memory is set to TRUE.
Definition at line 4335 of file m_neuro.f90.

8.10.3.95 perception_predation_risk_objective()

```
elemental real(srp) function the_neurobio::perception_predation_risk_objective (
            class(perception), intent(in) this )
```

Calculate the risk of **predation** as being **perceived** / **assessed** by this agent.

Note

It can be placed either to `PERCEPTION` (which might seem more logical as it is basically the *perception* of predation risk by the agent) or `APPRAISAL` level class. Here it is in the `APPRAISAL` because it is actually used here. It may also be safer here as we need completed perception objects and perception memory stack to assess the objective predation risk.

Returns

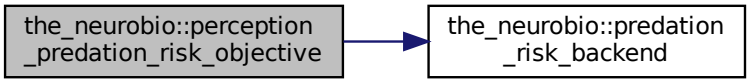
assessment of the predation risk based on both the perception object and its linked perceptual memory component.

8.10.3.95.1 Notable parameters **WEIGHT_DIRECT** is the relative weight given to the immediate perception of predators over the predators counts in the memory stack. Obtained from global parameters (`commondata::predation_risk_weight_immediate`).
MEM_WIND is the size of the memory window when assessing the predator risk, only this number of the latest elements from the memory stack is taken into account. So we further weight the direct threat over the background risk when making the decision.

Note

Note that we take into account the whole memory size (commondata::history_size_perception).

8.10.3.95.2 Implementation details Here we analyse the predator perception object and memory stack to get the **predation risk** perception value, that will be fed into the sigmoid function via `neuro_resp` at the next step. Perception of the predation risk is the weighted sum of the **total number of predators in the memory stack** and the **current count** of predators within the visual range of the agent. The actual calculatio is done by the comon backend that is used for objective and subjective assessment of risk: the_neurobio::predation_risk_backend().
Definition at line 4372 of file m_neuro.f90.
Here is the call graph for this function:



8.10.3.96 predation_risk_backend()

```
elemental real(srp) function the_neurobio::predation_risk_backend (
             integer, intent(in) pred_count,
             real(srp), intent(in) pred_memory_mean,
             real(srp), intent(in), optional weight_direct )
```

Simple computational backend for the risk of predation that is used in objective risk function the_neurobio::perception_predation_risk_o and the subjective risk function.

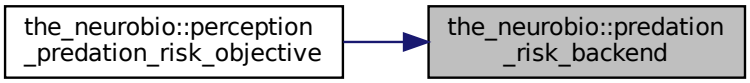
Parameters

in	*pred_count*	pred_count The number of predators in the current perception object. This is an estimate of the direct risk of predation r_d.
in	*pred_memory_mean*	pred_memory_mean The mean number of predators in the memory window. The size of the memory window is not set here. It is an estimate of the indirect risk of predation r_{id}.
in	*weight_direct*	weight_direct an optional weighting factor for the immediate risk (the number of predators in the current perception object), ω. If not provided, the default value is set by the commondata::predation_risk_weight_immediate parameter.

8.10.3.96.1 Implementation details First, check if the optional direct risk weighting factor ω) is provided as a dummy parameter. If not provided, use the default value that is set by the commondata::predation_risk_weight_immediate parameter.
Second, calculate the predation risk as a weighted sum of the direct risk (number of immediately perceived predators, r_d) and indirect risk (average number of predators in the memory, r_{id}):

$$R = r_d \cdot \omega + r_{id} \cdot (1 - \omega)$$

Definition at line 4414 of file m_neuro.f90.
Here is the caller graph for this function:



8.10.3.97 perception_components_attention_weights_init()

```
elemental subroutine the_neurobio::perception_components_attention_weights_init (
            class(percept_components_motiv), intent(inout) this,
            real(srp), intent(in), optional all_vals_fix,
            logical, intent(in), optional all_one,
            real(srp), intent(in), optional weight_light,
            real(srp), intent(in), optional weight_depth,
            real(srp), intent(in), optional weight_food_dir,
            real(srp), intent(in), optional weight_food_mem,
            real(srp), intent(in), optional weight_conspec,
            real(srp), intent(in), optional weight_pred_dir,
            real(srp), intent(in), optional weight_predator,
            real(srp), intent(in), optional weight_stomach,
            real(srp), intent(in), optional weight_bodymass,
            real(srp), intent(in), optional weight_energy,
            real(srp), intent(in), optional weight_age,
            real(srp), intent(in), optional weight_reprfac )
```

Initialise the attention components of the emotional state to their default parameter values. Attention sets weights to individual perceptual components when the overall weighted sum is calculated. The default weights are parameters defined in `COMMONDATA`.

Warning

The number and nature of the attention components is equal to the number of perceptual components, they agree 1 to 1.

Note

The perception weights `weight_` parameters are not passed as an array to (a) allow for elemental function, (b) allow disabling attention components at init when weights are not provided = set to zero.

The **precedence** order of the parameters `all_vals_fix`, `all_one` and then `weight_`s, i.e. if `all_vals_fix` is provided, all other are ignored (see `return` in if-present-test blocks).

Parameters

in	*all_vals_fix*	all_vals_fix Optional parameter setting all weights equal to a specific fixed value.
in	*all_one*	all_one Optional logical parameter setting all weights to 1.0, so the perceptual components go into unchanged form (weight=1) into the weighted sum (overall primary motivation value).
in	*weight_light*	Optional attention weights for specific perception components.

Note

If absent, set to **zero**.

Parameters

in	*weight_depth*	Optional attention weights for specific perception components.

Note

If absent, set to **zero**.

Parameters

in	*weight_food_dir*	Optional attention weights for specific perception components.

Note

If absent, set to **zero**.

Parameters

in	*weight_food_mem*	Optional attention weights for specific perception components.

Note

If absent, set to **zero**.

Parameters

in	*weight_conspec*	Optional attention weights for specific perception components.

Note

If absent, set to **zero**.

Parameters

in	*weight_pred_dir*	Optional attention weights for specific perception components.

Note

If absent, set to **zero**.

Parameters

in	*weight_predator*	Optional attention weights for specific perception components.

Note

If absent, set to **zero**.

Parameters

in	*weight_stomach*	Optional attention weights for specific perception components.

Note

If absent, set to **zero**.

Parameters

in	*weight_bodymass*	Optional attention weights for specific perception components.

Note

If absent, set to **zero**.

Parameters

in	*weight_energy*	Optional attention weights for specific perception components.

Note

If absent, set to **zero**.

Parameters

in	*weight_age*	Optional attention weights for specific perception components.

Note

If absent, set to **zero**.

Parameters

in	*weight_reprfac*	Optional attention weights for specific perception components.

Note

If absent, set to **zero**.

Local copies of the optional parameters.
If `all_vals_fix` is set, set all weights to this fixed value.

Note

We do not have option to set all values to an **array** to be able to have this procedure **elemental**.

Return after setting values.
If `all_one` is present and set to TRUE, init all attention weights to 1.0
Return after setting values.
Set individual attention weights
If nothing is provided, set attention weights from the dummy parameters of this procedure.
Definition at line 4477 of file m_neuro.f90.

8.10.3.98 perception_components_neuronal_response_init_set()

```
subroutine the_neurobio::perception_components_neuronal_response_init_set (
             class(percept_components_motiv), intent(inout) this,
             class(appraisal), intent(inout) this_agent,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_light,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_depth,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_food_dir,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_food_mem,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_conspec,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_pred_dir,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_predator,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_stomach,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_bodymass,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_energy,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_age,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_reprfac,
             real(srp), intent(in), optional param_gerror_cv_light,
             real(srp), intent(in), optional param_gerror_cv_depth,
             real(srp), intent(in), optional param_gerror_cv_food_dir,
             real(srp), intent(in), optional param_gerror_cv_food_mem,
             real(srp), intent(in), optional param_gerror_cv_conspec,
             real(srp), intent(in), optional param_gerror_cv_pred_dir,
             real(srp), intent(in), optional param_gerror_cv_predator,
             real(srp), intent(in), optional param_gerror_cv_stomach,
             real(srp), intent(in), optional param_gerror_cv_bodymass,
             real(srp), intent(in), optional param_gerror_cv_energy,
             real(srp), intent(in), optional param_gerror_cv_age,
             real(srp), intent(in), optional param_gerror_cv_reprfac,
             character(len=*), intent(in), optional param_gene_label_light,
             character(len=*), intent(in), optional param_gene_label_depth,
             character(len=*), intent(in), optional param_gene_label_food_dir,
             character(len=*), intent(in), optional param_gene_label_food_mem,
             character(len=*), intent(in), optional param_gene_label_conspec,
             character(len=*), intent(in), optional param_gene_label_pred_dir,
             character(len=*), intent(in), optional param_gene_label_predator,
```

```
character(len=*), intent(in), optional param_gene_label_stomach,
character(len=*), intent(in), optional param_gene_label_bodymass,
character(len=*), intent(in), optional param_gene_label_energy,
character(len=*), intent(in), optional param_gene_label_age,
character(len=*), intent(in), optional param_gene_label_reprfac )
```

Set and calculate individual perceptual components for **this** motivational state using the **neuronal response** function, for **this_agent**.

Note

The **this_agent** has intent [inout], so can be changed as a result of this procedure, gene labels are set for genes involved in the neuronal response.

TODO: huge parameter list- ugly coding, try to fix.

This procedure uses labelled if constructs with inline call of the neuronal response function `neuro_resp`, unlike this, the intent [in] procedure `perception_components_neuronal_response_calculate` uses inner subroutines.

Parameters

`in,out`	*this_agent*	[inout] this_agent The actor agent.
`in`	*param_gp_matrix_light*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

`in`	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
`in`	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
`in`	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food
`in`	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
`in`	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
`in`	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
`in`	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
`in`	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
`in`	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
`in`	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
`in`	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
`in`	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
`in`	*param_gp_matrix_depth*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_food_dir*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_food_mem*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_conspec*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_pred_dir*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_predator*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_stomach*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_bodymass*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_energy*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_age*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_reprfac*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gerror_cv_light*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light
in	*param_gerror_cv_depth*	coefficient of variation for depth
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_depth*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light
in	*param_gerror_cv_depth*	coefficient of variation for depth
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;

Parameters

in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_food_dir*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light
in	*param_gerror_cv_depth*	coefficient of variation for depth
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_food_mem*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light
in	*param_gerror_cv_depth*	coefficient of variation for depth
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_conspec*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light
in	*param_gerror_cv_depth*	coefficient of variation for depth
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;

Parameters

in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_pred_dir*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light
in	*param_gerror_cv_depth*	coefficient of variation for depth
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_predator*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light
in	*param_gerror_cv_depth*	coefficient of variation for depth
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_stomach*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light
in	*param_gerror_cv_depth*	coefficient of variation for depth
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;

Parameters

in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_bodymass*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light
in	*param_gerror_cv_depth*	coefficient of variation for depth
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_energy*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light
in	*param_gerror_cv_depth*	coefficient of variation for depth
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_age*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light
in	*param_gerror_cv_depth*	coefficient of variation for depth
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;

Parameters

in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_reprfac*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light
in	*param_gerror_cv_depth*	coefficient of variation for depth
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gene_label_light*	param_gene_label_light label for light;
in	*param_gene_label_depth*	label for depth;
in	*param_gene_label_food_dir*	label for direct food;
in	*param_gene_label_food_mem*	label for number of food items in memory;
in	*param_gene_label_conspec*	label for number of conspecific;
in	*param_gene_label_pred_dir*	label for direct predation risk;
in	*param_gene_label_predator*	label for number of predators;
in	*param_gene_label_stomach*	label for stomach contents;
in	*param_gene_label_bodymass*	label for body mass;
in	*param_gene_label_energy*	label for energy reserves;
in	*param_gene_label_age*	label for age;
in	*param_gene_label_reprfac*	label for reproductive factor;
in	*param_gene_label_depth*	param_gene_label_light label for light;
in	*param_gene_label_depth*	label for depth;
in	*param_gene_label_food_dir*	label for direct food;
in	*param_gene_label_food_mem*	label for number of food items in memory;
in	*param_gene_label_conspec*	label for number of conspecific;
in	*param_gene_label_pred_dir*	label for direct predation risk;
in	*param_gene_label_predator*	label for number of predators;
in	*param_gene_label_stomach*	label for stomach contents;
in	*param_gene_label_bodymass*	label for body mass;
in	*param_gene_label_energy*	label for energy reserves;
in	*param_gene_label_age*	label for age;
in	*param_gene_label_reprfac*	label for reproductive factor;
in	*param_gene_label_food_dir*	param_gene_label_light label for light;
in	*param_gene_label_depth*	label for depth;
in	*param_gene_label_food_dir*	label for direct food;
in	*param_gene_label_food_mem*	label for number of food items in memory;
in	*param_gene_label_conspec*	label for number of conspecific;

Parameters

in	*param_gene_label_pred_dir*	label for direct predation risk;
in	*param_gene_label_predator*	label for number of predators;
in	*param_gene_label_stomach*	label for stomach contents;
in	*param_gene_label_bodymass*	label for body mass;
in	*param_gene_label_energy*	label for energy reserves;
in	*param_gene_label_age*	label for age;
in	*param_gene_label_reprfac*	label for reproductive factor;
in	*param_gene_label_food_mem*	param_gene_label_light label for light;
in	*param_gene_label_depth*	label for depth;
in	*param_gene_label_food_dir*	label for direct food;
in	*param_gene_label_food_mem*	label for number of food items in memory;
in	*param_gene_label_conspec*	label for number of conspecific;
in	*param_gene_label_pred_dir*	label for direct predation risk;
in	*param_gene_label_predator*	label for number of predators;
in	*param_gene_label_stomach*	label for stomach contents;
in	*param_gene_label_bodymass*	label for body mass;
in	*param_gene_label_energy*	label for energy reserves;
in	*param_gene_label_age*	label for age;
in	*param_gene_label_reprfac*	label for reproductive factor;
in	*param_gene_label_conspec*	param_gene_label_light label for light;
in	*param_gene_label_depth*	label for depth;
in	*param_gene_label_food_dir*	label for direct food;
in	*param_gene_label_food_mem*	label for number of food items in memory;
in	*param_gene_label_conspec*	label for number of conspecific;
in	*param_gene_label_pred_dir*	label for direct predation risk;
in	*param_gene_label_predator*	label for number of predators;
in	*param_gene_label_stomach*	label for stomach contents;
in	*param_gene_label_bodymass*	label for body mass;
in	*param_gene_label_energy*	label for energy reserves;
in	*param_gene_label_age*	label for age;
in	*param_gene_label_reprfac*	label for reproductive factor;
in	*param_gene_label_pred_dir*	param_gene_label_light label for light;
in	*param_gene_label_depth*	label for depth;
in	*param_gene_label_food_dir*	label for direct food;
in	*param_gene_label_food_mem*	label for number of food items in memory;
in	*param_gene_label_conspec*	label for number of conspecific;
in	*param_gene_label_pred_dir*	label for direct predation risk;
in	*param_gene_label_predator*	label for number of predators;
in	*param_gene_label_stomach*	label for stomach contents;
in	*param_gene_label_bodymass*	label for body mass;
in	*param_gene_label_energy*	label for energy reserves;
in	*param_gene_label_age*	label for age;
in	*param_gene_label_reprfac*	label for reproductive factor;
in	*param_gene_label_predator*	param_gene_label_light label for light;
in	*param_gene_label_depth*	label for depth;
in	*param_gene_label_food_dir*	label for direct food;

Parameters

in	*param_gene_label_food_mem*	label for number of food items in memory;
in	*param_gene_label_conspec*	label for number of conspecific;
in	*param_gene_label_pred_dir*	label for direct predation risk;
in	*param_gene_label_predator*	label for number of predators;
in	*param_gene_label_stomach*	label for stomach contents;
in	*param_gene_label_bodymass*	label for body mass;
in	*param_gene_label_energy*	label for energy reserves;
in	*param_gene_label_age*	label for age;
in	*param_gene_label_reprfac*	label for reproductive factor;
in	*param_gene_label_stomach*	param_gene_label_light label for light;
in	*param_gene_label_depth*	label for depth;
in	*param_gene_label_food_dir*	label for direct food;
in	*param_gene_label_food_mem*	label for number of food items in memory;
in	*param_gene_label_conspec*	label for number of conspecific;
in	*param_gene_label_pred_dir*	label for direct predation risk;
in	*param_gene_label_predator*	label for number of predators;
in	*param_gene_label_stomach*	label for stomach contents;
in	*param_gene_label_bodymass*	label for body mass;
in	*param_gene_label_energy*	label for energy reserves;
in	*param_gene_label_age*	label for age;
in	*param_gene_label_reprfac*	label for reproductive factor;
in	*param_gene_label_bodymass*	param_gene_label_light label for light;
in	*param_gene_label_depth*	label for depth;
in	*param_gene_label_food_dir*	label for direct food;
in	*param_gene_label_food_mem*	label for number of food items in memory;
in	*param_gene_label_conspec*	label for number of conspecific;
in	*param_gene_label_pred_dir*	label for direct predation risk;
in	*param_gene_label_predator*	label for number of predators;
in	*param_gene_label_stomach*	label for stomach contents;
in	*param_gene_label_bodymass*	label for body mass;
in	*param_gene_label_energy*	label for energy reserves;
in	*param_gene_label_age*	label for age;
in	*param_gene_label_reprfac*	label for reproductive factor;
in	*param_gene_label_energy*	param_gene_label_light label for light;
in	*param_gene_label_depth*	label for depth;
in	*param_gene_label_food_dir*	label for direct food;
in	*param_gene_label_food_mem*	label for number of food items in memory;
in	*param_gene_label_conspec*	label for number of conspecific;
in	*param_gene_label_pred_dir*	label for direct predation risk;
in	*param_gene_label_predator*	label for number of predators;
in	*param_gene_label_stomach*	label for stomach contents;
in	*param_gene_label_bodymass*	label for body mass;
in	*param_gene_label_energy*	label for energy reserves;
in	*param_gene_label_age*	label for age;
in	*param_gene_label_reprfac*	label for reproductive factor;
in	*param_gene_label_age*	param_gene_label_light label for light;

Parameters

in	*param_gene_label_depth*	label for depth;
in	*param_gene_label_food_dir*	label for direct food;
in	*param_gene_label_food_mem*	label for number of food items in memory;
in	*param_gene_label_conspec*	label for number of conspecific;
in	*param_gene_label_pred_dir*	label for direct predation risk;
in	*param_gene_label_predator*	label for number of predators;
in	*param_gene_label_stomach*	label for stomach contents;
in	*param_gene_label_bodymass*	label for body mass;
in	*param_gene_label_energy*	label for energy reserves;
in	*param_gene_label_age*	label for age;
in	*param_gene_label_reprfac*	label for reproductive factor;
in	*param_gene_label_reprfac*	param_gene_label_light label for light;
in	*param_gene_label_depth*	label for depth;
in	*param_gene_label_food_dir*	label for direct food;
in	*param_gene_label_food_mem*	label for number of food items in memory;
in	*param_gene_label_conspec*	label for number of conspecific;
in	*param_gene_label_pred_dir*	label for direct predation risk;
in	*param_gene_label_predator*	label for number of predators;
in	*param_gene_label_stomach*	label for stomach contents;
in	*param_gene_label_bodymass*	label for body mass;
in	*param_gene_label_energy*	label for energy reserves;
in	*param_gene_label_age*	label for age;
in	*param_gene_label_reprfac*	label for reproductive factor;

Local copies of **genotype x phenotype** boolean matrices.
Local copies of the Gaussian perception error coefficients of variation.

8.10.3.98.1 Implementation notes We check input boolean G x P matrices and calculate the perceptual components of **this** motivation state only when the boolean matrix is provided as a parameter. Also check the corresponding variance/CV and reset to deterministic (variance zero) is not provided as a dummy parameter parameter.

Warning

There should be **exactly** as many `param_g_p_matrix` and `param_gerror_cv_light` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

- calculate the perceptual component for **light**.

Note

The function is almost the same as in `the_neurobio::appraisal` but **does** set the label so `the_↩agent` has **intent[inout]**.

- calculate the perceptual component for **depth**.
- calculate the perceptual component for **food_dir**.
- calculate the perceptual component for **food_mem**.
- calculate the perceptual component for **conspec**.
- calculate the perceptual component for **direct predation**.
- calculate the perceptual component for **predator**.

- calculate the perceptual component for **stomach**.
- calculate the perceptual component for **bodymass**.
- calculate the perceptual component for **energy**.
- calculate the perceptual component for **age**.
- calculate the perceptual component for **reproduct. factor**.

Definition at line 4668 of file m_neuro.f90.

8.10.3.99 perception_components_neuronal_response_calculate()

```
subroutine the_neurobio::perception_components_neuronal_response_calculate (
             class(percept_components_motiv), intent(inout) this,
             class(appraisal), intent(in) this_agent,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_light,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_depth,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_food_dir,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_food_mem,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_conspec,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_pred_dir,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_predator,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_stomach,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_bodymass,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_energy,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_age,
             logical, dimension(max_nalleles,n_chromosomes), intent(in), optional param_gp_↩
matrix_reprfac,
             real(srp), intent(in), optional param_gerror_cv_light,
             real(srp), intent(in), optional param_gerror_cv_depth,
             real(srp), intent(in), optional param_gerror_cv_food_dir,
             real(srp), intent(in), optional param_gerror_cv_food_mem,
             real(srp), intent(in), optional param_gerror_cv_conspec,
             real(srp), intent(in), optional param_gerror_cv_pred_dir,
             real(srp), intent(in), optional param_gerror_cv_predator,
             real(srp), intent(in), optional param_gerror_cv_stomach,
             real(srp), intent(in), optional param_gerror_cv_bodymass,
             real(srp), intent(in), optional param_gerror_cv_energy,
             real(srp), intent(in), optional param_gerror_cv_age,
             real(srp), intent(in), optional param_gerror_cv_reprfac,
             real(srp), intent(in), optional perception_override_light,
             real(srp), intent(in), optional perception_override_depth,
             real(srp), intent(in), optional perception_override_food_dir,
             real(srp), intent(in), optional perception_override_food_mem,
             real(srp), intent(in), optional perception_override_conspec,
             real(srp), intent(in), optional perception_override_pred_dir,
```

```
real(srp), intent(in), optional perception_override_predator,
real(srp), intent(in), optional perception_override_stomach,
real(srp), intent(in), optional perception_override_bodymass,
real(srp), intent(in), optional perception_override_energy,
real(srp), intent(in), optional perception_override_age,
real(srp), intent(in), optional perception_override_reprfac )
```

Calculate individual perceptual components for **this** motivational state using the **neuronal response** function, for an **this_agent**.

Note

The **this_agent** has intent [in], so is unchanged as a result of this procedure. Unlike the above intent [inout] procedure, this accepts optional perception parameters that override those stored in `this_agent` data structure. This is done for calculating representation **expectancies** from possible behaviour.

Parameters

in	*this_agent*	[inout] this_agent The actor agent.
in	*param_gp_matrix_light*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food;
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_depth*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food;
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_food_dir*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food;
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_food_mem*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food;
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_conspec*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food;
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_pred_dir*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food;
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_predator*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food;
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_stomach*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food;
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_bodymass*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food;
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_energy*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food;
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_age*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food;
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gp_matrix_reprfac*	### Boolean G x P matrices ### Input structure of the fixed parameters that define the boolean genotype x phenotype matrices for each perceptual component of motivational state defined in commondata.

Warning

There should be **exactly** as many `param_g_p_matrix` parameters as perceptual components for this motivation (`the_neurobio::percept_components_motiv`).

The dimensionality of the parameter arrays must be **exactly** the same as in commondata. This is why **assumed shape arrays** (:,:) are **not used** here.

Parameters

in	*param_gp_matrix_light*	boolean *G* x *P* matrix for light;
in	*param_gp_matrix_depth*	boolean *G* x *P* matrix for depth;
in	*param_gp_matrix_food_dir*	boolean *G* x *P* matrix for direct food;
in	*param_gp_matrix_food_mem*	boolean *G* x *P* matrix for number of food items in memory;
in	*param_gp_matrix_conspec*	boolean *G* x *P* matrix for number of conspecifics;
in	*param_gp_matrix_pred_dir*	boolean *G* x *P* matrix for direct predation risk;
in	*param_gp_matrix_predator*	boolean *G* x *P* matrix for number of predators;
in	*param_gp_matrix_stomach*	boolean *G* x *P* matrix for stomach contents;
in	*param_gp_matrix_bodymass*	boolean *G* x *P* matrix for body mass;
in	*param_gp_matrix_energy*	boolean *G* x *P* matrix for energy reserves;
in	*param_gp_matrix_age*	boolean *G* x *P* matrix for age;
in	*param_gp_matrix_reprfac*	boolean *G* x *P* matrix for reproductive factor.
in	*param_gerror_cv_light*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light;
in	*param_gerror_cv_depth*	coefficient of variation for depth;
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_depth*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light;
in	*param_gerror_cv_depth*	coefficient of variation for depth;
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_food_dir*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.

Parameters

in	*param_gerror_cv_light*	coefficient of variation for light;
in	*param_gerror_cv_depth*	coefficient of variation for depth;
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_food_mem*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light;
in	*param_gerror_cv_depth*	coefficient of variation for depth;
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_conspec*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light;
in	*param_gerror_cv_depth*	coefficient of variation for depth;
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_pred_dir*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.

Parameters

in	*param_gerror_cv_light*	coefficient of variation for light;
in	*param_gerror_cv_depth*	coefficient of variation for depth;
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_predator*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light;
in	*param_gerror_cv_depth*	coefficient of variation for depth;
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_stomach*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light;
in	*param_gerror_cv_depth*	coefficient of variation for depth;
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_bodymass*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.

Parameters

in	*param_gerror_cv_light*	coefficient of variation for light;
in	*param_gerror_cv_depth*	coefficient of variation for depth;
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_energy*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light;
in	*param_gerror_cv_depth*	coefficient of variation for depth;
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_age*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.
in	*param_gerror_cv_light*	coefficient of variation for light;
in	*param_gerror_cv_depth*	coefficient of variation for depth;
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*param_gerror_cv_reprfac*	### Coefficients of variation parameters ### Input structures that define the coefficient of variation for the Gaussian perception error parameters for each of the perceptual components. Normally they are also defined in commondata.

Parameters

in	*param_gerror_cv_light*	coefficient of variation for light;
in	*param_gerror_cv_depth*	coefficient of variation for depth;
in	*param_gerror_cv_food_dir*	coefficient of variation for direct food;
in	*param_gerror_cv_food_mem*	coefficient of variation for number of food items in memory;
in	*param_gerror_cv_conspec*	coefficient of variation for number of conspecific;
in	*param_gerror_cv_pred_dir*	coefficient of variation for direct predation risk;
in	*param_gerror_cv_predator*	coefficient of variation for number of predators;
in	*param_gerror_cv_stomach*	coefficient of variation for stomach contents;
in	*param_gerror_cv_bodymass*	coefficient of variation for body mass;
in	*param_gerror_cv_energy*	coefficient of variation for energy reserves;
in	*param_gerror_cv_age*	coefficient of variation for age;
in	*param_gerror_cv_reprfac*	coefficient of variation for reproductive factor.
in	*perception_override_light*	### Perception overrides (fake perceptions) ### Optional parameters to override the perception value of `this_agent` that will be passed through the neuronal response function. We need to be able to pass arbitrary perception values to neuronal response to assess expectancies of different behaviours for this agent.

Warning

Note that the data types of the perception override values **must** agree with the `init_val` parameter of the neuronal response function, i.e. be **real**. But the perception object accessor (get) function of the respective perception object (`PERCEPT_`) sometimes have integer type. In such cases use inline real conversion function **when calling this** procedure.

Parameters

in	*perception_override_light*	perception override for light;
in	*perception_override_depth*	perception override for depth;
in	*perception_override_food_dir*	perception override for direct food (convert from integer);
in	*perception_override_food_mem*	perception override for number of food items in memory;
in	*perception_override_conspec*	perception override for number of conspecific (convert from integer);
in	*perception_override_pred_dir*	perception override for direct predation risk;
in	*perception_override_predator*	perception override for number of predators;
in	*perception_override_stomach*	perception override for stomach contents;
in	*perception_override_bodymass*	perception override for body mass;
in	*perception_override_energy*	perception override for energy reserves;
in	*perception_override_age*	perception override for age (convert from integer);
in	*perception_override_reprfac*	perception override for reproductive factor.
in	*perception_override_depth*	### Perception overrides (fake perceptions) ### Optional parameters to override the perception value of `this_agent` that will be passed through the neuronal response function. We need to be able to pass arbitrary perception values to neuronal response to assess expectancies of different behaviours for this agent.

Warning

Note that the data types of the perception override values **must** agree with the `init_val` parameter of the neuronal response function, i.e. be **real**. But the perception object accessor (get) function of the respective perception object (`PERCEPT_`) sometimes have integer type. In such cases use inline real conversion function **when calling this** procedure.

Parameters

in	*perception_override_light*	perception override for light;
in	*perception_override_depth*	perception override for depth;
in	*perception_override_food_dir*	perception override for direct food (convert from integer);
in	*perception_override_food_mem*	perception override for number of food items in memory;
in	*perception_override_conspec*	perception override for number of conspecific (convert from integer);
in	*perception_override_pred_dir*	perception override for direct predation risk;
in	*perception_override_predator*	perception override for number of predators;
in	*perception_override_stomach*	perception override for stomach contents;
in	*perception_override_bodymass*	perception override for body mass;
in	*perception_override_energy*	perception override for energy reserves;
in	*perception_override_age*	perception override for age (convert from integer);
in	*perception_override_reprfac*	perception override for reproductive factor.
in	*perception_override_food_dir*	### Perception overrides (fake perceptions) ### Optional parameters to override the perception value of `this_agent` that will be passed through the neuronal response function. We need to be able to pass arbitrary perception values to neuronal response to assess expectancies of different behaviours for this agent.

Warning

Note that the data types of the perception override values **must** agree with the `init_val` parameter of the neuronal response function, i.e. be **real**. But the perception object accessor (get) function of the respective perception object (`PERCEPT_`) sometimes have integer type. In such cases use inline real conversion function **when calling this** procedure.

Parameters

in	*perception_override_light*	perception override for light;
in	*perception_override_depth*	perception override for depth;
in	*perception_override_food_dir*	perception override for direct food (convert from integer);
in	*perception_override_food_mem*	perception override for number of food items in memory;
in	*perception_override_conspec*	perception override for number of conspecific (convert from integer);
in	*perception_override_pred_dir*	perception override for direct predation risk;
in	*perception_override_predator*	perception override for number of predators;
in	*perception_override_stomach*	perception override for stomach contents;
in	*perception_override_bodymass*	perception override for body mass;
in	*perception_override_energy*	perception override for energy reserves;
in	*perception_override_age*	perception override for age (convert from integer);
in	*perception_override_reprfac*	perception override for reproductive factor.
in	*perception_override_food_mem*	### Perception overrides (fake perceptions) ### Optional parameters to override the perception value of `this_agent` that will be passed through the neuronal response function. We need to be able to pass arbitrary perception values to neuronal response to assess expectancies of different behaviours for this agent.

Warning

Note that the data types of the perception override values **must** agree with the `init_val` parameter of the neuronal response function, i.e. be **real**. But the perception object accessor (get) function of the respective perception object (`PERCEPT_`) sometimes have integer type. In such cases use inline real conversion function **when calling this** procedure.

Parameters

in	*perception_override_light*	perception override for light;
in	*perception_override_depth*	perception override for depth;
in	*perception_override_food_dir*	perception override for direct food (convert from integer);
in	*perception_override_food_mem*	perception override for number of food items in memory;
in	*perception_override_conspec*	perception override for number of conspecific (convert from integer);
in	*perception_override_pred_dir*	perception override for direct predation risk;
in	*perception_override_predator*	perception override for number of predators;
in	*perception_override_stomach*	perception override for stomach contents;
in	*perception_override_bodymass*	perception override for body mass;
in	*perception_override_energy*	perception override for energy reserves;
in	*perception_override_age*	perception override for age (convert from integer);
in	*perception_override_reprfac*	perception override for reproductive factor.
in	*perception_override_conspec*	### Perception overrides (fake perceptions) ### Optional parameters to override the perception value of `this_agent` that will be passed through the neuronal response function. We need to be able to pass arbitrary perception values to neuronal response to assess expectancies of different behaviours for this agent.

Warning

Note that the data types of the perception override values **must** agree with the `init_val` parameter of the neuronal response function, i.e. be **real**. But the perception object accessor (get) function of the respective perception object (`PERCEPT_`) sometimes have integer type. In such cases use inline real conversion function **when calling this** procedure.

Parameters

in	*perception_override_light*	perception override for light;
in	*perception_override_depth*	perception override for depth;
in	*perception_override_food_dir*	perception override for direct food (convert from integer);
in	*perception_override_food_mem*	perception override for number of food items in memory;
in	*perception_override_conspec*	perception override for number of conspecific (convert from integer);
in	*perception_override_pred_dir*	perception override for direct predation risk;
in	*perception_override_predator*	perception override for number of predators;
in	*perception_override_stomach*	perception override for stomach contents;
in	*perception_override_bodymass*	perception override for body mass;
in	*perception_override_energy*	perception override for energy reserves;
in	*perception_override_age*	perception override for age (convert from integer);
in	*perception_override_reprfac*	perception override for reproductive factor.
in	*perception_override_pred_dir*	### Perception overrides (fake perceptions) ### Optional parameters to override the perception value of `this_agent` that will be passed through the neuronal response function. We need to be able to pass arbitrary perception values to neuronal response to assess expectancies of different behaviours for this agent.

Warning

Note that the data types of the perception override values **must** agree with the `init_val` parameter of the neuronal response function, i.e. be **real**. But the perception object accessor (get) function of the respective perception object (`PERCEPT_`) sometimes have integer type. In such cases use inline real conversion function **when calling this** procedure.

Parameters

in	*perception_override_light*	perception override for light;
in	*perception_override_depth*	perception override for depth;
in	*perception_override_food_dir*	perception override for direct food (convert from integer);
in	*perception_override_food_mem*	perception override for number of food items in memory;
in	*perception_override_conspec*	perception override for number of conspecific (convert from integer);
in	*perception_override_pred_dir*	perception override for direct predation risk;
in	*perception_override_predator*	perception override for number of predators;
in	*perception_override_stomach*	perception override for stomach contents;
in	*perception_override_bodymass*	perception override for body mass;
in	*perception_override_energy*	perception override for energy reserves;
in	*perception_override_age*	perception override for age (convert from integer);
in	*perception_override_reprfac*	perception override for reproductive factor.
in	*perception_override_predator*	### Perception overrides (fake perceptions) ### Optional parameters to override the perception value of `this_agent` that will be passed through the neuronal response function. We need to be able to pass arbitrary perception values to neuronal response to assess expectancies of different behaviours for this agent.

Warning

Note that the data types of the perception override values **must** agree with the `init_val` parameter of the neuronal response function, i.e. be **real**. But the perception object accessor (get) function of the respective perception object (`PERCEPT_`) sometimes have integer type. In such cases use inline real conversion function **when calling this** procedure.

Parameters

in	*perception_override_light*	perception override for light;
in	*perception_override_depth*	perception override for depth;
in	*perception_override_food_dir*	perception override for direct food (convert from integer);
in	*perception_override_food_mem*	perception override for number of food items in memory;
in	*perception_override_conspec*	perception override for number of conspecific (convert from integer);
in	*perception_override_pred_dir*	perception override for direct predation risk;
in	*perception_override_predator*	perception override for number of predators;
in	*perception_override_stomach*	perception override for stomach contents;
in	*perception_override_bodymass*	perception override for body mass;
in	*perception_override_energy*	perception override for energy reserves;
in	*perception_override_age*	perception override for age (convert from integer);
in	*perception_override_reprfac*	perception override for reproductive factor.
in	*perception_override_stomach*	### Perception overrides (fake perceptions) ### Optional parameters to override the perception value of `this_agent` that will be passed through the neuronal response function. We need to be able to pass arbitrary perception values to neuronal response to assess expectancies of different behaviours for this agent.

Warning

Note that the data types of the perception override values **must** agree with the `init_val` parameter of the neuronal response function, i.e. be **real**. But the perception object accessor (get) function of the respective perception object (`PERCEPT_`) sometimes have integer type. In such cases use inline real conversion function **when calling this** procedure.

Parameters

in	*perception_override_light*	perception override for light;
in	*perception_override_depth*	perception override for depth;
in	*perception_override_food_dir*	perception override for direct food (convert from integer);
in	*perception_override_food_mem*	perception override for number of food items in memory;
in	*perception_override_conspec*	perception override for number of conspecific (convert from integer);
in	*perception_override_pred_dir*	perception override for direct predation risk;
in	*perception_override_predator*	perception override for number of predators;
in	*perception_override_stomach*	perception override for stomach contents;
in	*perception_override_bodymass*	perception override for body mass;
in	*perception_override_energy*	perception override for energy reserves;
in	*perception_override_age*	perception override for age (convert from integer);
in	*perception_override_reprfac*	perception override for reproductive factor.
in	*perception_override_bodymass*	### Perception overrides (fake perceptions) ### Optional parameters to override the perception value of `this_agent` that will be passed through the neuronal response function. We need to be able to pass arbitrary perception values to neuronal response to assess expectancies of different behaviours for this agent.

Warning

Note that the data types of the perception override values **must** agree with the `init_val` parameter of the neuronal response function, i.e. be **real**. But the perception object accessor (get) function of the respective perception object (`PERCEPT_`) sometimes have integer type. In such cases use inline real conversion function **when calling this** procedure.

Parameters

in	*perception_override_light*	perception override for light;
in	*perception_override_depth*	perception override for depth;
in	*perception_override_food_dir*	perception override for direct food (convert from integer);
in	*perception_override_food_mem*	perception override for number of food items in memory;
in	*perception_override_conspec*	perception override for number of conspecific (convert from integer);
in	*perception_override_pred_dir*	perception override for direct predation risk;
in	*perception_override_predator*	perception override for number of predators;
in	*perception_override_stomach*	perception override for stomach contents;
in	*perception_override_bodymass*	perception override for body mass;
in	*perception_override_energy*	perception override for energy reserves;
in	*perception_override_age*	perception override for age (convert from integer);
in	*perception_override_reprfac*	perception override for reproductive factor.
in	*perception_override_energy*	### Perception overrides (fake perceptions) ### Optional parameters to override the perception value of `this_agent` that will be passed through the neuronal response function. We need to be able to pass arbitrary perception values to neuronal response to assess expectancies of different behaviours for this agent.

Warning

Note that the data types of the perception override values **must** agree with the `init_val` parameter of the neuronal response function, i.e. be **real**. But the perception object accessor (get) function of the respective perception object (`PERCEPT_`) sometimes have integer type. In such cases use inline real conversion function **when calling this** procedure.

Parameters

in	*perception_override_light*	perception override for light;
in	*perception_override_depth*	perception override for depth;
in	*perception_override_food_dir*	perception override for direct food (convert from integer);
in	*perception_override_food_mem*	perception override for number of food items in memory;
in	*perception_override_conspec*	perception override for number of conspecific (convert from integer);
in	*perception_override_pred_dir*	perception override for direct predation risk;
in	*perception_override_predator*	perception override for number of predators;
in	*perception_override_stomach*	perception override for stomach contents;
in	*perception_override_bodymass*	perception override for body mass;
in	*perception_override_energy*	perception override for energy reserves;
in	*perception_override_age*	perception override for age (convert from integer);
in	*perception_override_reprfac*	perception override for reproductive factor.
in	*perception_override_age*	### Perception overrides (fake perceptions) ### Optional parameters to override the perception value of `this_agent` that will be passed through the neuronal response function. We need to be able to pass arbitrary perception values to neuronal response to assess expectancies of different behaviours for this agent.

Warning

Note that the data types of the perception override values **must** agree with the `init_val` parameter of the neuronal response function, i.e. be **real**. But the perception object accessor (get) function of the respective perception object (`PERCEPT_`) sometimes have integer type. In such cases use inline real conversion function **when calling this** procedure.

Parameters

in	*perception_override_light*	perception override for light;
in	*perception_override_depth*	perception override for depth;
in	*perception_override_food_dir*	perception override for direct food (convert from integer);
in	*perception_override_food_mem*	perception override for number of food items in memory;
in	*perception_override_conspec*	perception override for number of conspecific (convert from integer);
in	*perception_override_pred_dir*	perception override for direct predation risk;
in	*perception_override_predator*	perception override for number of predators;
in	*perception_override_stomach*	perception override for stomach contents;
in	*perception_override_bodymass*	perception override for body mass;
in	*perception_override_energy*	perception override for energy reserves;
in	*perception_override_age*	perception override for age (convert from integer);
in	*perception_override_reprfac*	perception override for reproductive factor.
in	*perception_override_reprfac*	### Perception overrides (fake perceptions) ### Optional parameters to override the perception value of `this_agent` that will be passed through the neuronal response function. We need to be able to pass arbitrary perception values to neuronal response to assess expectancies of different behaviours for this agent.

Warning

Note that the data types of the perception override values **must** agree with the `init_val` parameter of the neuronal response function, i.e. be **real**. But the perception object accessor (get) function of the respective perception object (`PERCEPT_`) sometimes have integer type. In such cases use inline real conversion function **when calling this** procedure.

Parameters

`in`	*perception_override_light*	perception override for light;
`in`	*perception_override_depth*	perception override for depth;
`in`	*perception_override_food_dir*	perception override for direct food (convert from integer);
`in`	*perception_override_food_mem*	perception override for number of food items in memory;
`in`	*perception_override_conspec*	perception override for number of conspecific (convert from integer);
`in`	*perception_override_pred_dir*	perception override for direct predation risk;
`in`	*perception_override_predator*	perception override for number of predators;
`in`	*perception_override_stomach*	perception override for stomach contents;
`in`	*perception_override_bodymass*	perception override for body mass;
`in`	*perception_override_energy*	perception override for energy reserves;
`in`	*perception_override_age*	perception override for age (convert from integer);
`in`	*perception_override_reprfac*	perception override for reproductive factor.

8.10.3.99.1 Implementation notes We check input boolean G x P matrices and calculate the perceptual components of **this** motivation state only when the boolean matrix is provided as a parameter. Also check the corresponding variance/CV and reset to deterministic (variance zero) is not provided as a dummy parameter parameter.

Warning

There should be **exactly** as many `param_g_p_matrix_` and `param_gerror_cv_` parameters as perceptual components for this motivation (the_neurobio::percept_components_motiv).

- calculate the perceptual component for **light**.
- calculate the perceptual component for **depth**.
- calculate the perceptual component for **food_dir**.
- calculate the perceptual component for **food_mem**.
- calculate the perceptual component for **conspec**.
- calculate the perceptual component for **direct predation**.
- calculate the perceptual component for **predator**.
- calculate the perceptual component for **stomach**.
- calculate the perceptual component for **bodymass**.
- calculate the perceptual component for **energy**.
- calculate the perceptual component for **age**.
- calculate the perceptual component for **reproductive factor**.

Definition at line 5233 of file m_neuro.f90.

8.10.3.100 state_motivation_light_get()

```
elemental real(srp) function the_neurobio::state_motivation_light_get (
            class(state_motivation_base), intent(in) this )
```

Standard "get" function for the state neuronal **light** effect component.

Returns

function value: light

Definition at line 5753 of file m_neuro.f90.

8.10.3.101 state_motivation_depth_get()

```
elemental real(srp) function the_neurobio::state_motivation_depth_get (
            class(state_motivation_base), intent(in) this )
```

Standard "get" function for the state neuronal **depth** effect component.

Returns

function value: depth

Definition at line 5765 of file m_neuro.f90.

8.10.3.102 state_motivation_food_dir_get()

```
elemental real(srp) function the_neurobio::state_motivation_food_dir_get (
            class(state_motivation_base), intent(in) this )
```

Standard "get" function for the state neuronal **directly seen food** effect component.

Returns

function value: food

Definition at line 5777 of file m_neuro.f90.

8.10.3.103 state_motivation_food_mem_get()

```
elemental real(srp) function the_neurobio::state_motivation_food_mem_get (
            class(state_motivation_base), intent(in) this )
```

Standard "get" function for the state neuronal **food items from past memory** effect component.

Returns

function value: food

Definition at line 5789 of file m_neuro.f90.

8.10.3.104 state_motivation_conspec_get()

```
elemental real(srp) function the_neurobio::state_motivation_conspec_get (
            class(state_motivation_base), intent(in) this )
```

Standard "get" function for the state neuronal **conspecifics** effect component.

Returns

function value: conspecifics

Definition at line 5801 of file m_neuro.f90.

8.10.3.105 state_motivation_pred_dir_get()

```
elemental real(srp) function the_neurobio::state_motivation_pred_dir_get (
            class(state_motivation_base), intent(in) this )
```

Standard "get" function for the state neuronal **direct predation** effect component.

Returns

function value: predators

Definition at line 5813 of file m_neuro.f90.

8.10.3.106 state_motivation_predator_get()

```
elemental real(srp) function the_neurobio::state_motivation_predator_get (
            class(state_motivation_base), intent(in) this )
```

Standard "get" function for the state neuronal **predators** effect component.

Returns

function value: predators

Definition at line 5825 of file m_neuro.f90.

8.10.3.107 state_motivation_stomach_get()

```
elemental real(srp) function the_neurobio::state_motivation_stomach_get (
            class(state_motivation_base), intent(in) this )
```

Standard "get" function for the state neuronal **stomach** effect component.

Returns

function value: stomach

Definition at line 5837 of file m_neuro.f90.

8.10.3.108 state_motivation_bodymass_get()

```
elemental real(srp) function the_neurobio::state_motivation_bodymass_get (
            class(state_motivation_base), intent(in) this )
```

Standard "get" function for the state neuronal **body mass** effect component.

Returns

function value: body mass

Definition at line 5849 of file m_neuro.f90.

8.10.3.109 state_motivation_energy_get()

```
elemental real(srp) function the_neurobio::state_motivation_energy_get (
            class(state_motivation_base), intent(in) this )
```

Standard "get" function for the state neuronal **energy reserves** effect component.

Returns

function value: energy reserves

Definition at line 5861 of file m_neuro.f90.

8.10.3.110 state_motivation_age_get()

```
elemental real(srp) function the_neurobio::state_motivation_age_get (
            class(state_motivation_base), intent(in) this )
```

Standard "get" function for the state neuronal **age** effect component.

Returns

function value: age

Definition at line 5873 of file m_neuro.f90.

8.10.3.111 state_motivation_reprfac_get()

```
elemental real(srp) function the_neurobio::state_motivation_reprfac_get (
            class(state_motivation_base), intent(in) this )
```

Standard "get" function for the state neuronal **reproductive factor** effect component.

Returns

function value: age

Definition at line 5885 of file m_neuro.f90.

8.10.3.112 state_motivation_motivation_prim_get()

```
elemental real(srp) function the_neurobio::state_motivation_motivation_prim_get (
            class(state_motivation_base), intent(in) this )
```

Standard "get" function for the root state, get the overall **primary motivation value** (before modulation).

Returns

function value: age

Definition at line 5897 of file m_neuro.f90.

8.10.3.113 state_motivation_motivation_get()

```
elemental real(srp) function the_neurobio::state_motivation_motivation_get (
            class(state_motivation_base), intent(in) this )
```

Standard "get" function for the root state, get the overall **final motivation value** (after modulation).

Returns

function value: age

Definition at line 5909 of file m_neuro.f90.

8.10.3.114 state_motivation_is_dominant_get()

```
elemental logical function the_neurobio::state_motivation_is_dominant_get (
            class(state_motivation_base), intent(in) this )
```

Check if the root state is the dominant state in GOS.

Returns

TRUE if this motivational state is **dominant** in the GOS, and FALSE otherwise.

Definition at line 5920 of file m_neuro.f90.

8.10.3.115 state_motivation_fixed_label_get()

```
elemental character(len=label_length) function the_neurobio::state_motivation_fixed_label_get
(
            class(state_motivation_base), intent(in) this )
```

Get the fixed label for this motivational state. Note that the label is fixed and cannot be changed.

Returns

Returns the fixed label for this motivation state.

Definition at line 5934 of file m_neuro.f90.

8.10.3.116 state_motivation_attention_weights_transfer()

```
pure subroutine the_neurobio::state_motivation_attention_weights_transfer (
            class(state_motivation_base), intent(inout) this,
            class(state_motivation_base), intent(in) copy_from )
```

Transfer attention weights between two motivation state components. The main use of this subroutine would be to transfer attention from the actor agent's main motivation's attention component to the behaviour's GOS expectancy object.

Note

Note that the procedure `behaviour_root_attention_weights_transfer` which does this is not using this procedure and transfers objects directly.

Definition at line 5952 of file m_neuro.f90.

8.10.3.117 perception_component_maxval()

```
elemental real(srp) function the_neurobio::perception_component_maxval (
            class(percept_components_motiv), intent(in) this )
```

Calculate the **maximum** value over all the perceptual components.

Returns

the maximum value among all the perceptual components.

Definition at line 5976 of file m_neuro.f90.

8.10.3.118 state_motivation_percept_maxval()

```
elemental real(srp) function the_neurobio::state_motivation_percept_maxval (
            class(state_motivation_base), intent(in) this )
```

Calculate the **maximum** value over all the perceptual components of this motivational state component.

Note

Used in `motivation_primary_calc` procedure.

Returns

the maximum value among all the perceptual components.

Definition at line 6002 of file m_neuro.f90.

8.10.3.119 state_motivation_calculate_prim()

```
elemental real(srp) function the_neurobio::state_motivation_calculate_prim (
            class(state_motivation_base), intent(in) this,
            real(srp), intent(in), optional maxvalue )
```

Calculate the level of **primary motivation** for this **specific** emotional state **component**.

Note

Used in `motivation_primary_calc` procedure.

Parameters

in	*maxvalue*	The maximum value across all appraisal perception components, needed to standardise and rescale the latter to the range 0:1 before they are summed up.

Returns

The value of the primary motivation for this motivation component.

Local parameters defining 0.0 and 1.0 for rescale.
Local copy of optional `maxvalue`.
Normally we **rescale** all values within the perceptual motivation components coming from the appraisal level into a [0..1] range within the agent, so that they are comparable across the motivations. To do this we need the maximum perception value **over all perception objects**: `maxvalue`. Normally `maxvalue` is an input parameter taking account of all motivation all state components. But if it is not provided, we calculate local maximum for **this** motivational component only.
Calculate the primary motivation for this motivational state by summing up (**averaging**) all the perceptual components for this motivation weighted by their respective attention weights; components are ∗∗rescaled∗ from the potential global range 0:maxvalue to the range 0:1.

Note

`maxvalue` should normally be the maximum value for **all** available motivation states, not just this. TODO: make maxvalue a structure reflecting motivational components.

Definition at line 6016 of file m_neuro.f90.

8.10.3.120 perception_component_motivation_init_zero()

```
elemental subroutine the_neurobio::perception_component_motivation_init_zero (
            class(percept_components_motiv), intent(inout) this )
```

Initialise perception components for a motivation state object.
Definition at line 6099 of file m_neuro.f90.

8.10.3.121 state_hunger_zero()

```
elemental subroutine the_neurobio::state_hunger_zero (
            class(state_hunger), intent(inout) this )
```

Init and cleanup **hunger** motivation object. The only difference from the base root STATE_MOTIVATION_BASE is that it sets unique label.
Definition at line 6120 of file m_neuro.f90.

8.10.3.122 state_fear_defence_zero()

```
elemental subroutine the_neurobio::state_fear_defence_zero (
            class(state_fear_defence), intent(inout) this )
```

Init and cleanup **fear state** motivation object. The only difference from the base root STATE_MOTIVATION_BASE is that it sets unique label.
Definition at line 6152 of file m_neuro.f90.

8.10.3.123 state_reproduce_zero()

```
elemental subroutine the_neurobio::state_reproduce_zero (
            class(state_reproduce), intent(inout) this )
```

Init and cleanup **reproductive** motivation object. The only difference from the base root STATE_MOTIVATION_↩ BASE is that it sets unique label.
Definition at line 6184 of file m_neuro.f90.

8.10.3.124 motivation_init_all_zero()

```
elemental subroutine the_neurobio::motivation_init_all_zero (
            class(motivation), intent(inout) this )
```

Init the expectancy components to a zero state.
Expectancy components.
Also set the private and fixed "number of motivational states" constant, we obviously have 3 motivations.
Definition at line 6214 of file m_neuro.f90.

8.10.3.125 motivation_reset_gos_indicators()

```
elemental subroutine the_neurobio::motivation_reset_gos_indicators (
            class(motivation), intent(inout) this )
```

Reset all GOS indicators for this motivation object.
Reset dominant status to FALSE for all motivational states.
Definition at line 6230 of file m_neuro.f90.

8.10.3.126 motivation_max_perception_calc()

```
elemental real(srp) function the_neurobio::motivation_max_perception_calc (
            class(motivation), intent(in) this )
```

Calculate maximum value of the perception components across all motivations.

Returns

Returns the maximum value of the perception components across all motivations.

Definition at line 6243 of file m_neuro.f90.

8.10.3.127 motivation_return_final_as_vector()

```
pure real(srp) function, dimension(:), allocatable the_neurobio::motivation_return_final_as_↩
vector (
            class(motivation), intent(in) this )
```

Return the vector of final motivation values for all motivational state components.
Definition at line 6260 of file m_neuro.f90.

8.10.3.128 motivation_maximum_value_motivation_finl()

```
elemental real(srp) function the_neurobio::motivation_maximum_value_motivation_finl (
            class(motivation), intent(in) this )
```

Calculate the maximum value of the final motivations across all motivational state components.

Parameters

in	*this*	this self

Returns

Maximum final motivation.

An equivalent "manual" form not using `finals` function : maxvalue = maxval([thishungermotivation_finl, & thisfear_defencemotivation_finl, & thisreproductionmotivation_finl])
Definition at line 6274 of file m_neuro.f90.

8.10.3.129 motivation_val_is_maximum_value_motivation_finl()

```
elemental logical function the_neurobio::motivation_val_is_maximum_value_motivation_finl (
             class(motivation), intent(in) this,
             real(srp), intent(in) test_value )
```

Checks if the test value is the maximum **final** motivation value across all motivational state components.

Note

This is a scalar form, inputs a single scalar value for testing.

Definition at line 6290 of file m_neuro.f90.

8.10.3.130 motivation_val_is_maximum_value_motivation_finl_o()

```
elemental logical function the_neurobio::motivation_val_is_maximum_value_motivation_finl_o (
             class(motivation), intent(in) this,
             class(state_motivation_base), intent(in) test_motivation )
```

Checks if the test value is the maximum **final** motivation value across all motivational state components.

Note

This is object form, inputs a whole motivation state object

Definition at line 6309 of file m_neuro.f90.

8.10.3.131 motivation_primary_sum_components()

```
elemental subroutine the_neurobio::motivation_primary_sum_components (
             class(motivation), intent(inout) this,
             real(srp), intent(in), optional max_val )
```

Calculate the **primary motivations** from motivation-specific perception appraisal components. The **primary motivations** are motivation values before the modulation takes place.

Parameters

in	*max_val*	max_val optional parameter that sets the maximum perception value for rescaling all perceptions to a common currency.

Note

Needed to standardise and rescale the appraisal perception components to the range 0:1 before they are summed up.

8.10.3.131.1 Implementation notes

- Rescale all values within the perceptual motivation components coming from the appraisal level into a [0..1] range within the agent, so that they are comparable across the motivations. To do this we need the maximum perception value overall perception objects: `appmaxval`.

If the maximum rescale perception is provided as a parameter, use it.

- If the parameter value is not provided, calculate maximum rescale perceptions from the currently available perception components of each specific motivational state.
- Calculate each of the motivations by summing up all state perceptual components for a particular motivation that are rescaled from the potential global range 0:appmaxval to the range 0:1. This is done calling the the_neurobio::state_motivation_base::motivation_calculate() method for each of the motivational states.

Definition at line 6328 of file m_neuro.f90.

8.10.3.132 motivation_modulation_absent()

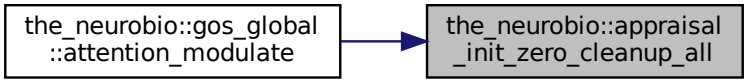
```
elemental subroutine the_neurobio::motivation_modulation_absent (
            class(motivation), intent(inout) this )
```

Produce **modulation** of the primary motivations, that result in the **final motivation** values (`_finl`). In this subroutine, **modulation is absent**, so the final motivation values are equal to the primary motivations.
Here the final motivations are just equal to their primary values
Definition at line 6385 of file m_neuro.f90.

8.10.3.133 appraisal_init_zero_cleanup_all()

```
elemental subroutine, private the_neurobio::appraisal_init_zero_cleanup_all (
            class(appraisal), intent(inout) this )  [private]
```

Initialise and cleanup all appraisal object components and sub-objects.
Init and clean all motivational components.
Also cleanup the emotional memory stack.
Definition at line 6448 of file m_neuro.f90.
Here is the caller graph for this function:



8.10.3.134 appraisal_agent_set_dead()

```
elemental subroutine the_neurobio::appraisal_agent_set_dead (
            class(appraisal), intent(inout) this )
```

Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function.
The `dies` method is implemented at the following levels of the agent object hierarchy (upper overrides the lower level):

- the_genome::individual_genome::dies();
- the_neurobio::appraisal::dies();

- the_neurobio::gos_global::dies();
- the_individual::individual_agent::dies().

Note

This method overrides the the_genome::individual_genome::dies() method, nullifying all reproductive and neurobiological and behavioural objects.

The `dies` method is implemented at the gos_global to allow "cleaning" of all neurobiological objects when `dies` is called when performing the behaviours upwards in the object hierarchy.

- Set the agent "dead";
- emptify reproduction objects;
- emptify all neurobiological objects.

Definition at line 6477 of file m_neuro.f90.

8.10.3.135 appraisal_perceptual_comps_motiv_neur_response_calculate()

```
subroutine the_neurobio::appraisal_perceptual_comps_motiv_neur_response_calculate (
             class(appraisal), intent(inout) this )
```

Get the perceptual components of all motivational states by passing perceptions via the neuronal response function.

Warning

Here we use the intent[inout] procedure that **does change** the actor agent: sets the labels for the genes. This procedure, therefore is used only for initialisation and not in prediction.

8.10.3.135.1 Implementation notes Call the neuronal response initialisation procedure the_neurobio::percept_components_motiv for all the motivation states:

- the_neurobio::state_hunger
- the_neurobio::state_fear_defence
- the_neurobio::state_reproduce

Definition at line 6493 of file m_neuro.f90.

8.10.3.136 appraisal_primary_motivations_calculate()

```
elemental subroutine the_neurobio::appraisal_primary_motivations_calculate (
             class(appraisal), intent(inout) this,
             real(srp), intent(in), optional rescale_max_motivation )
```

Calculate primary motivations from perceptual components of each motivation state.

Parameters

in	*rescale_max_motivation*	rescale_max_motivation maximum motivation value for rescaling all motivational components for comparison across all motivation and perceptual components and behaviour units.

8.10.3.136.1 Implementation notes

- Check if the maximum motivation value for rescale is provided as a parameter.

Check if global maximum motivation across all behaviours and perceptual components is provided for rescaling.

- If not, use local maximum value for this behaviour only.

Finally, the primary motivation values are calculated using the the_neurobio::motivation::motivation_primary_calc() method.
Definition at line 6644 of file m_neuro.f90.

8.10.3.137 appraisal_motivation_modulation_non_genetic()

```
subroutine the_neurobio::appraisal_motivation_modulation_non_genetic (
            class(appraisal), intent(inout) this,
            logical, intent(in), optional no_modulation )
```

Produce **modulation** of the primary motivations, that result in the **final motivation** values (_finl). Modulation here is non-genetic and involves a fixed transformation of the primary motivation values.

Parameters

in	*no_modulation*	no_genetic_modulation chooses if genetic modulation is calculated at all, if set to TRUE, then genetic modulation is **not** calculated and the final motivational values are just equal to the primary motivations.

8.10.3.137.1 Notable variables and parameters

- AGE_ARRAY_ABSCISSA is the interpolation grid abscissa for the weighting factor applied to reproductive motivation. It is defined by the parameter commondata::reprod_modulation_devel_abscissa.

AGE_ARRAY_ORDINATE is the interpolation grid ordinate. Its first and last values are set as 0.0 and 1.0, and the middle is defined by the parameter commondata::reprod_modulation_devel_w2.

```
htintrpl.exe [ 7000, 8555, 11666 ] [ 0, 0.10, 1.0 ]
```

8.10.3.137.2 Implementation notes First, *initialise* the **final motivation** values from the **no modulation** method the_neurobio::motivation::modulation_none().
Then check if developmental or genetic (or any other) modulation is disabled by the parameters commondata::modulation_appraisal_di
Check if no_genetic_modulation parameter is set to TRUE and if yes, return without no further processing.

8.10.3.137.2.1 Developmental modulation of reproductive factor Reproductive factor the_hormones::hormones::reproductive_f is accumulated by the sex hormone level whenever the agent is growing. Such accumulation can increase motivation for reproduction. However, reproduction is not possible in young and small agents. Therefore, this procedure implements a developmental modulation of the reproductive factor: reproductive motivation the_neurobio::state_reproduce is weighted out while the agent does not reach a target body length and age. This weighting is defined by nonlinear interpolation using the abscissa array AGE_ARRAY_ABSCISSA and ordinate AGE_ARRAY_ORDINATE. Such weighting, thus, allows non-zero reproductive motivation only when the agent reaches the age exceeding the first abscissa value AGE_ARRAY_ABSCISSA, age $>$ *L/2*, as here the weighting factor exceeds zero. Furthermore, when the age of the agent exceeds the last value of AGE_ARRAY_ABSCISSA, the weighting factor is equal to 1.0, so reproductive motivation is not limited any more.
Interpolation plots can be saved in the debug mode using this plotting command: commondata::debug_interpolate_plot_save().
Definition at line 6686 of file m_neuro.f90.

8.10.3.138 appraisal_motivation_modulation_genetic()

```
subroutine the_neurobio::appraisal_motivation_modulation_genetic (
            class(appraisal), intent(inout) this,
            logical, intent(in), optional no_genetic_modulation )
```

Produce **modulation** of the primary motivations, that result in the **final motivation** values (_finl). Modulation involves effects of such characteristics of the agent as body mass and age on the primary motivations (hunger, active and passive avoidance and reproduction) mediated by the genome effects. Here the genome determines the coefficients that set the degree of the influence of the agent's characteristics on the motivations.

Parameters

in	*no_genetic_modulation*	no_genetic_modulation chooses if genetic modulation is calculated at all, if set to TRUE, then genetic modulation is **not** calculated and the final motivational values are just equal to the primary motivations.

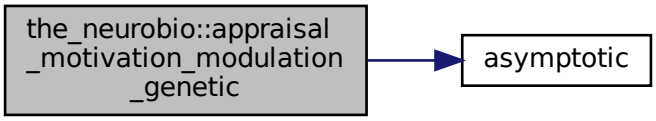
8.10.3.138.1 Implementation notes First, *initialise* the **final motivation** values from the **no modulation** method the_neurobio::motivation::modulation_none().
Then check if developmental or genetic (or any other) modulation is disabled by the parameters commondata::modulation_appraisal_di
Check if `no_genetic_modulation` parameter is set to TRUE and if yes, return without no further processing.
Sex modulation of the **reproduction** motivation state for **male**.

- First, use the sigmoid function and genome to set the phenotypic value of the **gamma** modulation coefficient that mediates the effect of the agent's sex on reproductive motivation.
- Second, add the modulation factor to the actual motivation value. If the agent is male, then its reproductive motivation is increased by an additive component that is an asymptotic() function of the genome based `modulation_gamma` parameter. The maximum modulatory increase ever possible is the double value of the raw primary motivation.

Sex modulation of the **reproduction** motivation state for **female**.

- First, use the sigmoid function and genome to set the phenotypic value of the **gamma** modulation coefficient that mediates the effect of the agent's sex on reproductive motivation.
- Second, add the modulation factor to the actual motivation value. If the agent is male, then its reproductive motivation is increased by an additive component that is an asymptotic() function of the genome based `modulation_gamma` parameter. The maximum modulatory increase ever possible is the double value of the raw primary motivation.

The values are logged in thedebug mode.
Definition at line 6786 of file m_neuro.f90.
Here is the call graph for this function:



8.10.3.139 appraisal_add_final_motivations_memory()

```
elemental subroutine the_neurobio::appraisal_add_final_motivations_memory (
            class(appraisal), intent(inout) this )
```

Add individual final emotional state components into the emotional memory stack. This is a wrapper to the the_neurobio::memory_emotional::add_to_memory method.
Definition at line 6934 of file m_neuro.f90.

8.10.3.140 reproduce_do_probability_reproduction_calc()

```
real(srp) function the_neurobio::reproduce_do_probability_reproduction_calc (
            class(appraisal), intent(in) this,
            real(srp), intent(in), optional weight_baseline,
            logical, intent(in), optional allow_immature )
```

Calculate the instantaneous probability of successful reproduction.

Note

Note that this function is bound to class the_neurobio::appraisal rather than the_neurobio::reproduce. Probability of successful reproduction is a dynamic property of this agent that depends on the nearby conspecifics and their sex and size/mass.

Parameters

in	*weight_baseline*	weight_baseline is the weighting factor for the baseline probability of successful reproduction φ (see details below).
in	*allow_immature*	allow_immature a logical switch that allows calculation (non-zero probability) if the agent is not ready to reproduction as determined by the the_body::reproduction::is_ready_reproduce() method. Normally, immature agents (for which this method returns FALSE) have zero probability of reproduction. The default is FALSE, i.e. not to allow reproduction to immature agents.

Returns

instantaneous probability of successful reproduction.

8.10.3.140.1 Implementation details The probability of successful reproduction depends on the number of conspecifics of the same and the opposite sex within the `this` agent's visual range. So the starting point here is the number of conspecifics within the current conspecifics perception object.
First, determine if the hormonal system of the agent is ready for reproduction using the_body::reproduction::is_ready_reproduce(). If this agent is not ready to reproduce, a zero probability of reproduction returned. However, if the optional parameter `allow_immature` is explicitly set to TRUE, this check is not done and the probability of reproduction is calculated as follows.
Second, determine if there are any conspecifics in the perception, if there are no, reproduction is impossible. Return straight away zero probability in such a case.
Second, extract the number of conspecifics `n_conspecifics_perception` from the perception object.
Also, initialise the number of same- and opposite-sex conspecifics (integer counters) as well as the total mass of same-sex conspecifics (real) to zero.
Third, determine how many of the conspecifics in perception have the same and the opposite sex. Calculate total mass of same sex conspecifics.
Additionally, check if the number of opposite sex agents is zero. in such a case zero probability of reproduction is obviously returned.
Fourth, calculate the **baseline probability** of reproduction. This probability is proportional to the proportions of the same- and opposite-sex agents within the visual range.

$$p_0 = \frac{N_{os}}{1 + N_{ss}}; 0 \leq p_0 \leq 1,$$

where N_{os} is the number of the opposite-sex agents, N_{ss} is the number of same-sex agents. We also adjust the baseline probability of successful reproduction by a parameter factor φ, so that this probability never reaches 1:

$$p_0 = \frac{N_{os}}{1 + N_{ss}} \cdot \varphi$$

For example, if there is only one agent of the opposite sex and no same-sex in proximity the baseline probability of reproduction is 1/(1+0) = 1.0 (note that the this agent also adds to the same-sex count, hence "1+..."). If there are 3 opposite-sex agents and 3 same-sex agents, the baseline probability is calculated as 3/(1+3) = 0.75. This doesn't take account of the φ multiplier factor.

Fifth, to get the final successful reproduction probability, the baseline value p_0 is multiplied by a function Φ that depends on the relative body mass of the `this` agent with respect to all the *same-sex* agents in proximity.

$$p_{rep} = p_0 \cdot \Phi(\Delta\overline{m_i}), 0 \leq p_{rep} \leq 1,$$

where p_{rep} is the final probability of successful reproduction. This is done to model direct within-sex competition for mates. Therefore, if the `this` agent is smaller than all the other same-sex agents here, the probability of successful reproduction significantly reduces. On the other hand, if the agent is larger than all the others, this probability would increase. The form of the Φ function is calculated on the bases of the *ratio* of the `this` agent body mass to the average body mass of all same sex agents within the visual range:

$$\Delta\overline{m_i} = \frac{M}{\overline{m_i}}.$$

Note that if there are *no same sex agents* (i.e. intra-sexual competition is absent) the probability of reproduction takes the baseline value p_0 :

$$p_{rep} = p_0.$$

No debug interpolation plot is produced in such a degenerate case.
The $\Phi(\Delta\overline{m_i})$ function itself is obtained from a nonlinear interpolation of grid values defined by the parameter arrays commondata::probability_reproduction_delta_mass_abscissa and commondata::probability_reproduction_delta_mass_ordinate.

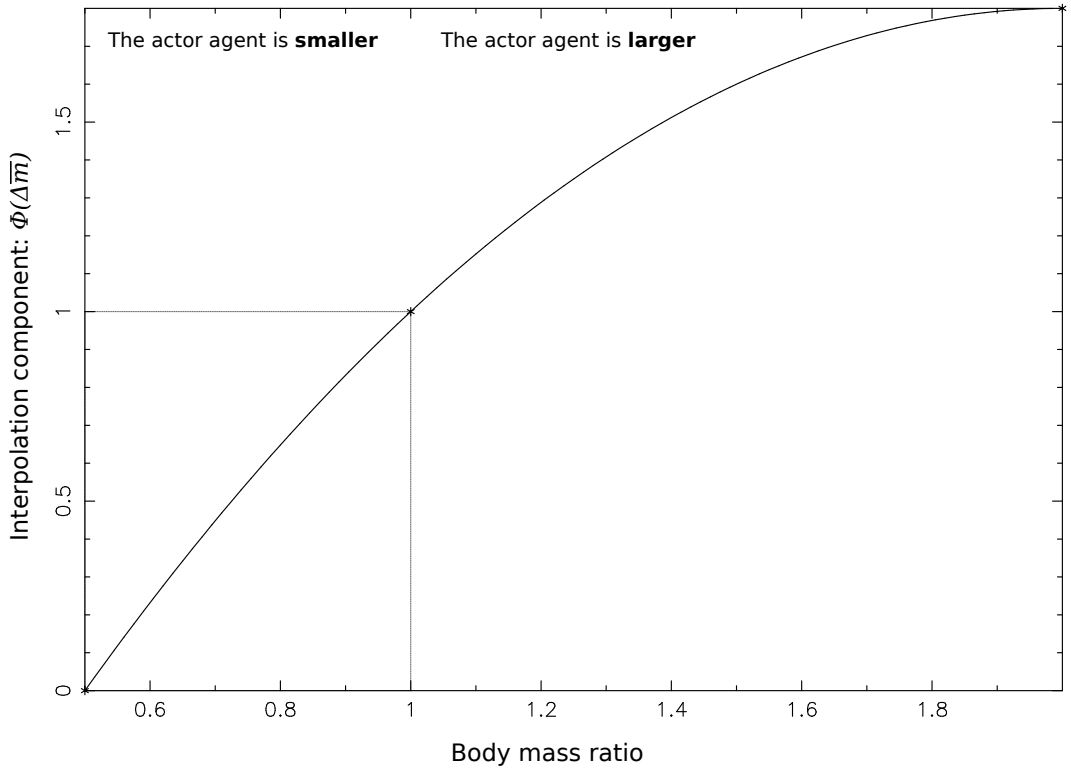


Figure 8.25 Relationship between body mass ratio and probability of reproduction

So the **final reproduction probability** value is obtained by multiplication of the baseline value by the Φ function.
The final probability of reproduction value is limited to lie within the range $0 \leq p_{rep} \leq 1 \cdot \varphi$.
Interpolation plots can be saved in the debug mode using this plotting command: commondata::debug_interpolate_plot_save().
Definition at line 6950 of file m_neuro.f90.

8.10.3.141 reproduction_success_stochast()

```
logical function the_neurobio::reproduction_success_stochast (
            class(appraisal), intent(in) this,
            real(srp), intent(in), optional prob )
```

Determine a stochastic outcome of **this** agent reproduction. Returns TRUE if the agent has reproduced successfully.

Parameters

in	*prob*	optional fixed probability of reproduction to override.

Returns

TRUE if reproduction is successful.

Warning

This function cannot be made elemental/pure due to random number call.

8.10.3.141.1 Implementation details Check if `prob` is present, if not, the probability of reproduction is calculated based on the perception objects of the actor agent `this` using the `probability_reproduction()` method.
Determine the reproduction success stochastically based on the probability of repriduction (`prob_here`) value.
Definition at line 7194 of file m_neuro.f90.

8.10.3.142 emotional_memory_add_to_stack()

```
elemental subroutine the_neurobio::emotional_memory_add_to_stack (
            class(memory_emotional), intent(inout) this,
            real(srp), intent(in) v_hunger,
            real(srp), intent(in) v_defence_fear,
            real(srp), intent(in) v_reproduction,
            character(*), intent(in), optional v_gos_label,
            real(srp), intent(in), optional v_gos_arousal,
            integer, intent(in), optional v_gos_repeated )
```

Add emotional components into the memory stack.

Parameters

in	*v_hunger*	The parameters of the subroutine are the actual values that are added to the emotional memory stack arrays.
in	*v_hunger*	value for hunger;
in	*v_defence_fear*	value for fear state;
in	*v_reproduction*	value for reproduction;
in	*v_gos_label*	value for GOS label;
in	*v_gos_arousal*	value for GOS arousal value;
in	*v_gos_repeated*	value for repeated counter for GOS.

Each of the memory stack components corresponds to the respective dummy parameter. These arrays are updated at each step (mandatory procedure arguments):

- v_hunger
- v_defence_fear
- v_reproduction

However, GOS parameters are optional and updated only if provided for invocation of this method (optional arguments):

- v_gos_label;
- v_gos_arousal;
- v_gos_repeated.

Definition at line 7227 of file m_neuro.f90.

8.10.3.143 emotional_memory_add_gos_to_stack()

```
elemental subroutine the_neurobio::emotional_memory_add_gos_to_stack (
            class(memory_emotional), intent(inout) this,
            character(*), intent(in), optional v_gos_label,
            real(srp), intent(in), optional v_gos_arousal,
            integer, intent(in), optional v_gos_repeated )
```

Add the current GOS label or/and arousal value and/or arousal repeat count into the emotional memory stack.

Parameters

in	*v_gos_label*	v_gos_label Text label for the current GOS.
in	*v_gos_arousal*	v_gos_arousal The maximum motivation (arousal) value for the current GOS.
in	*v_gos_repeated*	v_gos_repeated

8.10.3.143.1 Implementation notes GOS label is added to the memory stack.
GOS arousal is added to the memory stack.
The GOS repeated counter (gos_repeated) is added to the memory stack.
Definition at line 7275 of file m_neuro.f90.

8.10.3.144 emotional_memory_cleanup_stack()

```
elemental subroutine the_neurobio::emotional_memory_cleanup_stack (
            class(memory_emotional), intent(inout) this )
```

Cleanup and destroy the emotional memory stack.
cleanup procedure uses whole array assignment to the commondata::missing values.
Definition at line 7302 of file m_neuro.f90.

8.10.3.145 emotional_memory_hunger_get_mean()

```
elemental real(srp) function the_neurobio::emotional_memory_hunger_get_mean (
            class(memory_emotional), intent(in) this,
            integer, intent(in), optional last )
```

Get the average value of the hunger motivation state within the whole emotional memory stack.

Returns

Total count of predators in the memory stack.

Parameters

in	*last*	last Limit to only this number of latest components in the history.

8.10.3.145.1 Implementation notes

- Check if we are given the parameter requesting the latest history size. If the `last` parameter is absent or bigger than the array size, get the whole stack array.

Calculate the average excluding missing values (masked) within the subarray of interest.
Definition at line 7320 of file m_neuro.f90.

8.10.3.146 emotional_memory_actve_avoid_get_mean()

```
elemental real(srp) function the_neurobio::emotional_memory_actve_avoid_get_mean (
            class(memory_emotional), intent(in) this,
            integer, intent(in), optional last )
```

Get the average value of the fear state motivation state within the whole emotional memory stack.

Returns

Total count of predators in the memory stack.

Parameters

in	*last*	last Limit to only this number of latest components in the history.

8.10.3.146.1 Implementation notes

- Check if we are given the parameter requesting the latest history size. if the `last` parameter is absent or bigger than the array size, get the whole stack array.

Calculate the average excluding missing values (masked) within the subarray of interest.
Definition at line 7361 of file m_neuro.f90.

8.10.3.147 emotional_memory_reproduct_get_mean()

```
elemental real(srp) function the_neurobio::emotional_memory_reproduct_get_mean (
            class(memory_emotional), intent(in) this,
            integer, intent(in), optional last )
```

Get the average value of the reproductive motivation state within the whole emotional memory stack.

Returns

Total count of predators in the memory stack.

Parameters

in	*last*	last Limit to only this number of latest components in the history.

8.10.3.147.1 Implementation notes

- Check if we are given the parameter requesting the latest history size. if the `last` parameter is absent or bigger than the array size, get the whole stack array.

Calculate the average excluding missing values (masked) within the subarray of interest.
Definition at line 7403 of file m_neuro.f90.

8.10.3.148 emotional_memory_arousal_mean()

```
elemental real(srp) function the_neurobio::emotional_memory_arousal_mean (
            class(memory_emotional), intent(in) this,
            integer, intent(in), optional last )
```

Get the average value of the GOS arousal within the whole emotional memory stack.

Returns

Total count of predators in the memory stack.

Parameters

in	*last*	last Limit to only this number of latest components in the history.

8.10.3.148.1 Implementation notes

- Check if we are given the parameter requesting the latest history size. If the `last` parameter is absent or bigger than the array size, get the whole stack array.

Calculate the average excluding missing values (masked) within the subarray of interest.
Definition at line 7445 of file m_neuro.f90.

8.10.3.149 gos_find_global_state()

```
subroutine the_neurobio::gos_find_global_state (
             class(gos_global), intent(inout) this )
```

Find and set the **Global Organismic State (GOS)** of the agent based on the various available motivation values. The motivation values linked with the different stimuli compete with the current GOS and among themselves.

8.10.3.149.1 General principle The GOS competition threshold is a function of the current GOS arousal level: if it is very low, it would be very difficult to switch to a different GOS. However, if the current GOS has a high arousal, then switching to a competing motivation is relatively easy: a very small motivational surplus is enough for winning the competition with the current GOS.

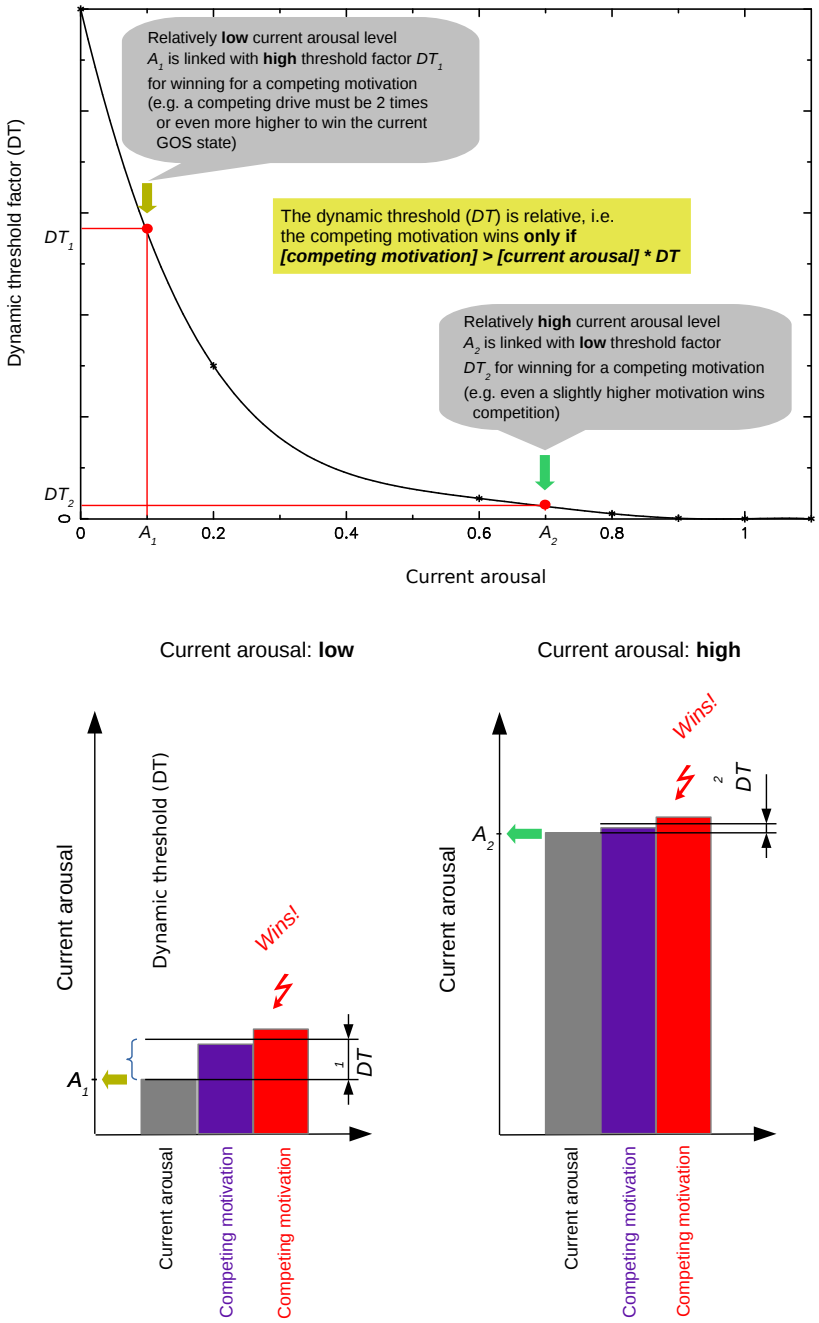


Figure 8.26 Global organismcs state

Note

GOS generation is a little changed in the new generation model.

1. We try to avoid constant switching of the GOS by requiring that the difference between motivational components should exceed some threshold value, if it does not, retain old GOS. So minor fluctuations in the stimulus field are ignored. Threshold is a dynamic parameter, so can also be zero.
2. The threshold is inversely related to the absolute value of the motivations compared, when the motivations are low, the threshold is big, when their values are approaching 1, the threshold approaches zero.

So motivations have relatively little effects.

8.10.3.149.2 Implementation details

8.10.3.149.2.1 Notable class data members **Public attribute of the `GOS_GLOBAL` class: `gos_arousal`** keeps the current level of the GOS arousal (*A*, see below). If GOS does switch as a result of competition with the other motivational states, it gets the value of its *winning* (maximum) motivation, if GOS does not switch as a result of competition, the `gos_arousal` value dissipates spontaneously to a lower value and the `gos_repeated` attribute of `GOS_GLOBAL` gets the successive number of repetitions of the same out of competition GOS state.

8.10.3.149.2.2 Notable local variables **Local variable: `arousal_new`** is the maximum level of motivation among all new incoming motivations *A*. It is this motivation value that competes with the current GOS arousal value (*G* the `gos_arousal` public attribute of the `GOS_GLOBAL` class).
Local variable `gos_dthreshold` is a dynamic threshold factor for GOS change Δ (see below). It determines the threshold that a new competing motivation has to exceed to win the competition with the previous (and still current up to this point) motivation.

8.10.3.149.2.3 GOS competition The GOS competition threshold is a function of the current GOS arousal level G: if it is very low, we need a relatively high competing motivation to win competition, if it is high then very small difference is enough. The global organismic state will switch to a competing state only if its maximum motivation A exceeds the current GOS's arousal level G by more than Δ units of G:

$$A - G > \Delta \cdot G.$$

Here the Δ threshold factor is set by a nonparametric function that is calculated from nonlinear interpolation of the grid values:

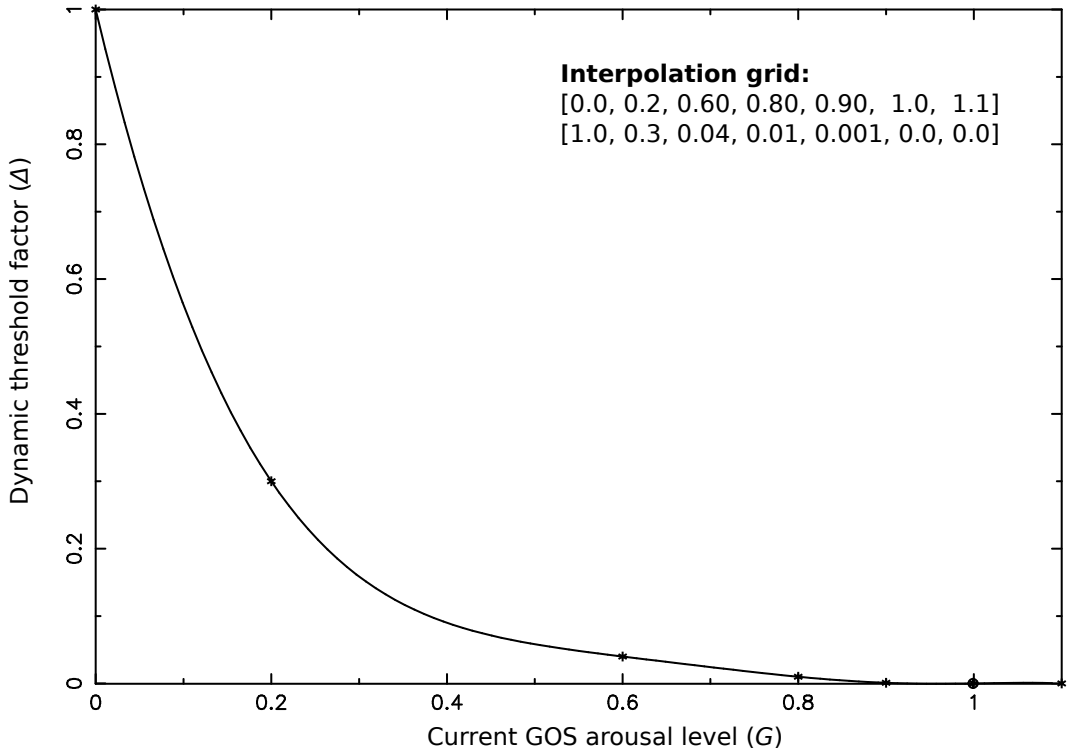


Figure 8.27 GOS competition threshold factor

So if the agent currently has a low GOS arousal *G*=0.1, it requires a competing state to be at least *A*=0.155 to win (with Δ =0.55, 0.155 = 0.1 + 0.1 $*$ 0.55). However, if the agent has a high GOS motivation *G*=0.8, almost any exceeding motivation ($>$0.808) will win. The actual value of the nonparametric interpolation function are obtained by nonlinear interpolation of the grid values defined by the `MOTIVATION_COMPET_THRESHOLD_CURVE_`↩ parameter arrays.

Note

In this implementation, the exact **type** of the competing motivation is not considered in the GOS competition procedure. For example, The current hunger GOS competes with all motivations, including itself. Consider an agent that is starving and has a high level of hunger GOS. At each step this hunger competes with all other motivations, including hunger. If hunger continues to increase, still, at many time steps, the new level of hunger can outcompete the current hunger and GOS switches from hunger to ... hunger. There can also be situations when current GOS hunger wins competition from all other motivations several times, dissipates and then is outcompeted by hunger again, that would lead to a relatively long streak of the same GOS. This mechanism would preclude switching out of (and losing) a continuously high but still appropriate motivational state.

The interpolation plots are saved in the debug mode to a disk file using an external command by the `commondata::debug_interpolate_plot_save()` procedure.

Warning

Enabling plotting can produce a **huge** number of plots and should normally be disabled.

Once the dynamic threshold is calculated, we can compare each of the competing motivation levels with the current arousal. If the maximum value of these motivations exceeds the current arousal by more than the threshold Δ factor, the GOS switches to the new motivation. If not, we are still left with the previous GOS.

Threshold not exceeded If the maximum competing motivation does not exceed the threshold, we are left with the old GOS. However, we reduce the current arousal spontaneously using a simple linear or some non-linear dissipation pattern using the %gos_repeated parameter that sets the number of repeated occurrences of the same (current) GOS. First, increment GOS repeat counter.
And spontaneously decrease, **dissipate**, the current arousal level. Spontaneous dissipation of arousal is implemented by multiplying the current level by a factor within the range [0.0..1.0] that can depend on the number of times this GOS is repeated.

Note

Note that the dissipation function is local to this procedure. `arousal_decrease_factor_fixed` = fixed value `arousal_decrease_factor_nonpar` = nonlinear, nonparametric, based on nonlinear interpolation. @plot `aha_gos_arousal_dissipation.svg`

Can use either `arousal_decrease_factor_fixed` or `arousal_decrease_factor_nonpar`.

Threshold is exceeded If the maximum competing motivation exceeds the threshold, we get to a **new GOS**. That is, the **highest** among the competing motivations defines the new GOS.

Note

Note that `gos_repeated` is initialised to 1.0 at `gos_reset`.

8.10.3.149.3 Check hunger Reset all motivations to *non-dominant*.
Set new GOS for hunger...

8.10.3.149.4 Check fear_defence Reset all motivations to *non-dominant*.
Set new GOS for fear_defence...

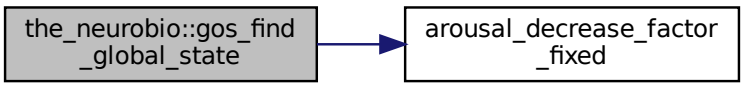
8.10.3.149.5 Check reproduction Reset all motivations to *non-dominant*.
Set new GOS for reproduction...

8.10.3.149.5.1 Other finalising procedures Add the current GOS parameters to the emotional memory stack

Note

Note that the memory stack arrays are defined in APPRAISAL and cleaned/init in `init_appraisal`

Finally recalculate the attention weights for all the states' perception components using attention_modulate(). The dominant GOS state will now get its default attention weights whereas all non-dominant states will get modulated values, i.e. values recalculated from a non-linear interpolation based **attention modulation curve**.
Definition at line 7508 of file m_neuro.f90.
Here is the call graph for this function:



8.10.3.150 gos_init_zero_state()

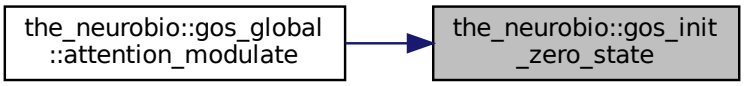
```
elemental subroutine, private the_neurobio::gos_init_zero_state (
            class(gos_global), intent(inout) this )  [private]
```

Initialise GOS engine components to a zero state. The values are set to commondata::missing, commondata::unknown, string to "undefined".

Note

the GOS arousal value is initialised to commondata::missing, which is a big negative value. Therefore, any competing motivation initially wins in the the_neurobio::gos_find_global_state() procedure. There seems to be no sense initialising arousal to 0.0.

Definition at line 7787 of file m_neuro.f90.
Here is the caller graph for this function:



8.10.3.151 gos_agent_set_dead()

```
elemental subroutine the_neurobio::gos_agent_set_dead (
            class(gos_global), intent(inout) this )
```

Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function.
The `dies` method is implemented at the following levels of the agent object hierarchy (upper overrides the lower level):

- the_genome::individual_genome::dies();

- the_neurobio::appraisal::dies();
- the_neurobio::gos_global::dies();
- the_individual::individual_agent::dies().

Note

This method overrides the the_genome::individual_genome::dies() method, nullifying all reproductive and neurobiological and behavioural objects.

The `dies` method is implemented at the gos_global to allow "cleaning" of all neurobiological objects when `dies` is called when performing the behaviours upwards in the object hierarchy.

- Set the agent "dead";
- emptify reproduction objects;
- emptify all neurobiological objects.

Definition at line 7818 of file m_neuro.f90.

8.10.3.152 gos_reset_motivations_non_dominant()

```
elemental subroutine the_neurobio::gos_reset_motivations_non_dominant (
            class(gos_global), intent(inout) this )
```

Reset all motivation states as *not* dominant with respect to the GOS.

Note

This subroutine is used in the_neurobio::gos_find_global_state().

8.10.3.152.1 Implementation notes Reset dominant status to FALSE for all motivational states calling the the_neurobio::motivation::gos_ind_reset().
Also reset the number of GOS repeated occurrences to 1.
Definition at line 7832 of file m_neuro.f90.

8.10.3.153 gos_global_get_label()

```
elemental character(len=label_length) function the_neurobio::gos_global_get_label (
            class(gos_global), intent(in) this )
```

Get the current global organismic state (GOS).

Returns

Global organismic state label.

Check which of the currently implemented motivational state components (`STATE_`) has the **dominant** flag. Can call motivation-type-bound function %is_dominant().

Note

Only one component can be "dominant".

Definition at line 7847 of file m_neuro.f90.

8.10.3.154 gos_get_arousal_level()

```
elemental real(srp) function the_neurobio::gos_get_arousal_level (
            class(gos_global), intent(in) this )
```

Get the overall level of arousal. Arousal is the current level of the dominant motivation that has brought about the current GOS at the previous time step.
Definition at line 7882 of file m_neuro.f90.

8.10.3.155 gos_attention_modulate_weights()

```
subroutine the_neurobio::gos_attention_modulate_weights (
            class(gos_global), intent(inout) this )
```

Modulate the attention weights to suppress all perceptions alternative to the current GOS. This is done using the attention modulation interpolation curve.

Warning

This subroutine is called from within `gos_find()` and should normally **not** be called separately.

8.10.3.155.1 Implementation details

8.10.3.155.1.1 Overview Each of the perceptions is weighted by an attention factor. The attention factor is in turn modulated (weighted) by the current Global Organismic State (GOS). When the current arousal is relatively high, all irrelevant perceptions are effectively filtered out (weighted by near-zero) and do not (largely) contribute to the GOS at the *next* time step. For example, the agent just does not "see" food items when it is in a high fear state.

Each of the **perceptions** is weighted by an **attention** factor. The attention factor is In turn **modulated** (weighted) by the current Global Organismic State **(GOS).** When the current arousal is relatively high, all irrelevant perceptions are effectively filtered out (weighted by near-zero) and do not (largely) contribute to the GOS at the next time step. For example, the agent does not "see" food items in a high fear state.

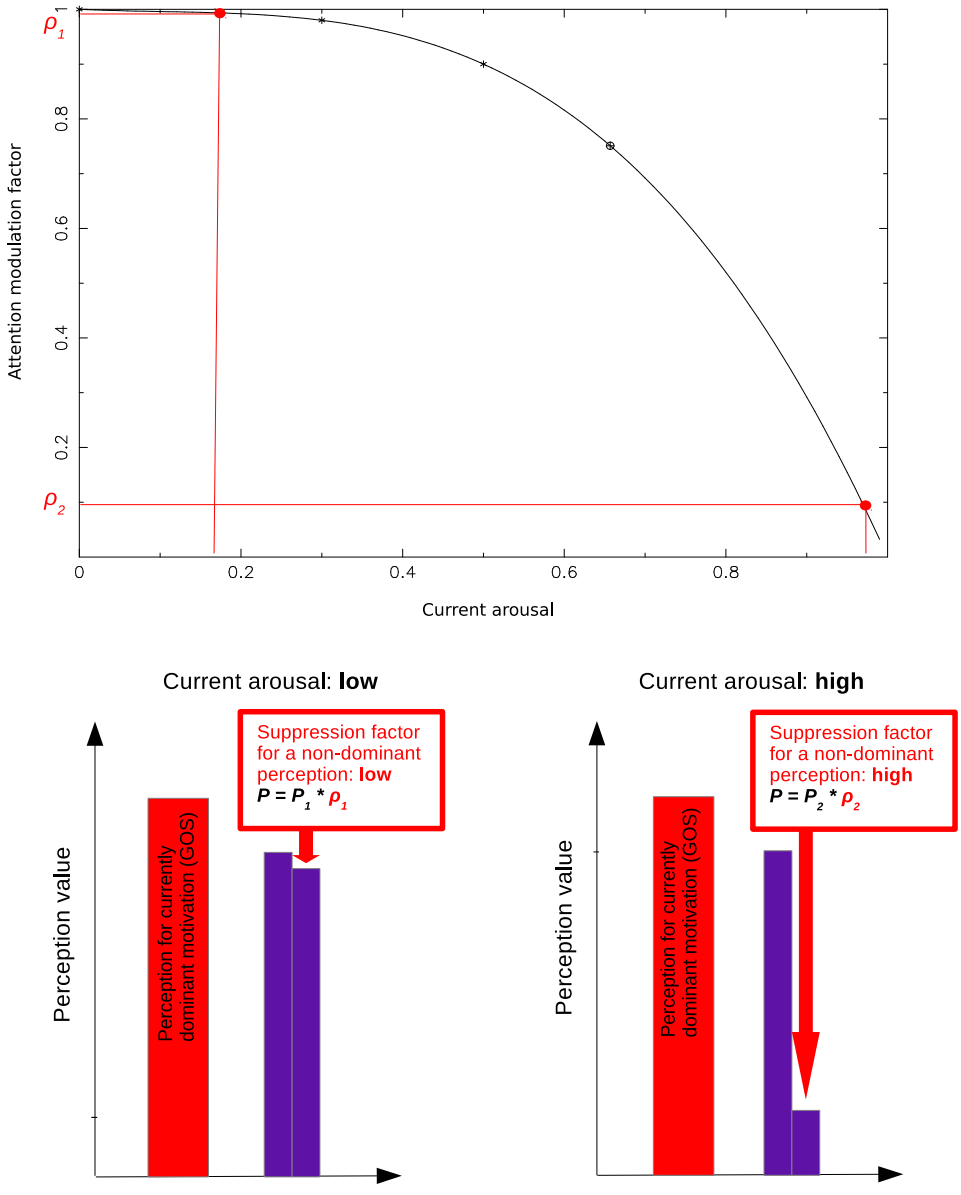


Figure 8.28 Perception weights

Thus, perception is weighted by the attention suppression factor separately for each motivational (emotional) state according to the scheme below:

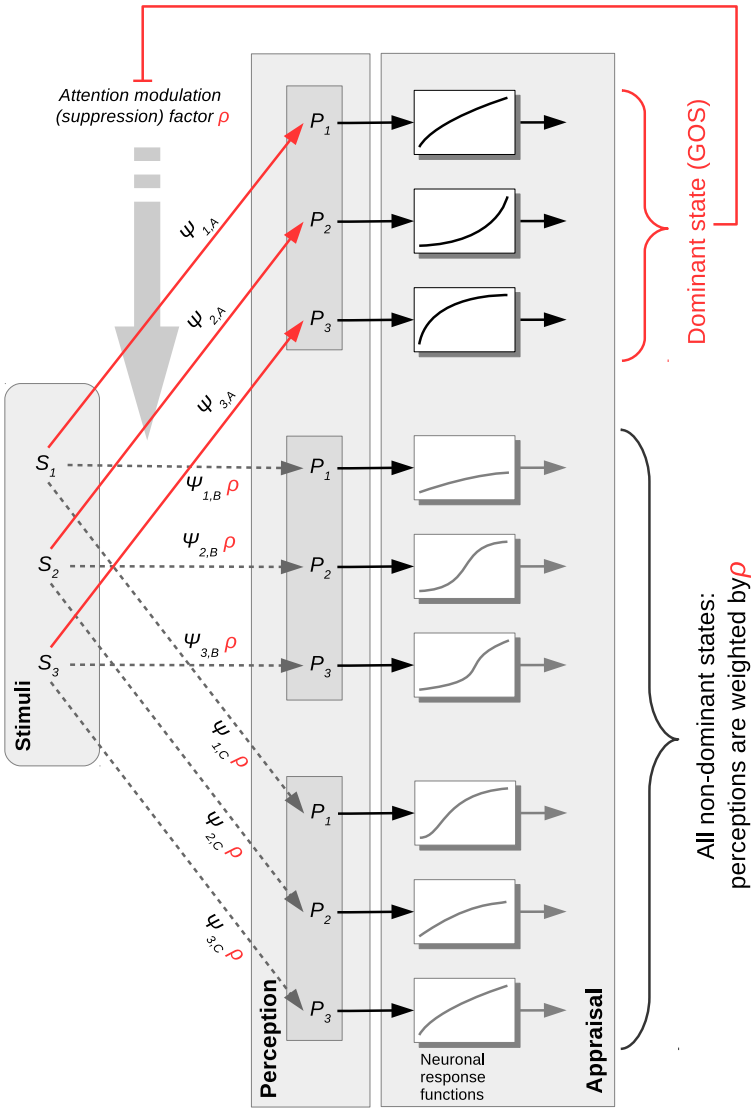


Figure 8.29 Attention suppression

Also see Cognitive architecture section.

8.10.3.155.1.2 Specific details **First**, we calculate the attention weight given to all non-dominant perceptions via nonlinear interpolation. Interpolation is based on the grid defined by two parameters: `ATTENTION_MODULATION_CURVE_ABSCISSA` and `ATTENTION_MODULATION_CURVE_ORDINATE`.

Note

Interpolation plot can be produced using this command, assuming the plotting tools are installed on the system.

```
htintrpl.exe [0.0, 0.3, 0.5, 1.0] [1.0, 0.98, 0.9, 0.0] [2]
```

Interpolation plots can be saved in the debug mode using this plotting command: commondata::debug_interpolate_plot_save().

Warning

Involves **huge** number of plots, should normally be disabled.

Second, we reset the attention weights for the **dominant GOS state** to their **default** parameter values whereas for all other states, to the **recalculated** `percept_w` modulated/weighted value. The the_neurobio::percept_components_motiv::attention_init() method is used to adjust the attention weights.
Definition at line 7899 of file m_neuro.f90.

8.10.3.156 perception_food_items_below_calculate()

```
elemental integer function the_neurobio::perception_food_items_below_calculate (
            class(perception), intent(in) this )
```

Calculate the number of food items in the perception object that are located **below** the actor agent.

Returns

The number of food items within the perception object that are located below (under) the actor agent.

8.10.3.156.1 Implementation details First, initialise the counter to zero.
Then, check if the agent has any food items in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one food item in the perception object. Calculate food items within the perception that are *below* the agent.
Definition at line 8112 of file m_neuro.f90.

8.10.3.157 perception_food_items_below_horiz_calculate()

```
elemental integer function the_neurobio::perception_food_items_below_horiz_calculate (
            class(perception), intent(in) this,
            real(srp), intent(in) hz_lower,
            real(srp), intent(in) hz_upper )
```

Calculate the number of food items in the perception object that are located **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z+hz_lower, z+hz_upper].

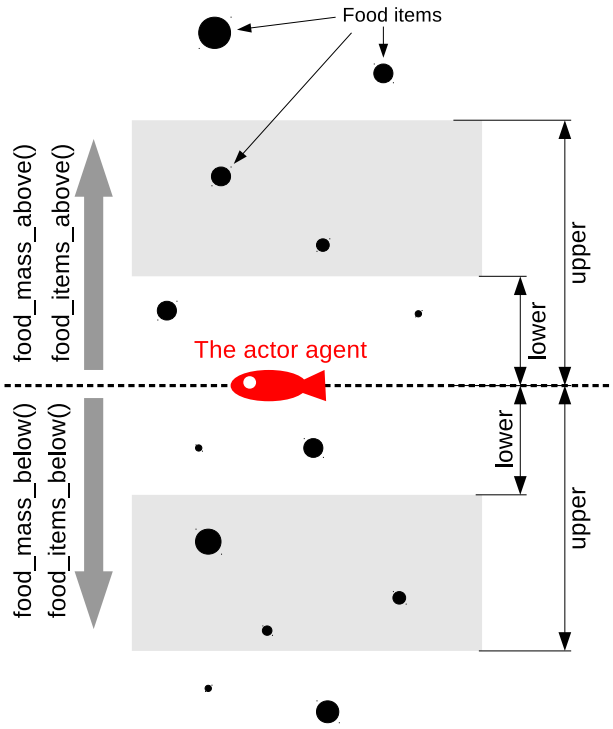


Figure 8.30 Calculation of food items above and below

Parameters

in	*hz_lower*	hz_lower The upper limit for the vertical horizon
in	*hz_upper*	hz_upper The lower limit for the vertical horizon

Returns

The number of food items within the perception object that are located below (under) the actor agent.

8.10.3.157.1 Implementation details First, initialise the counter to zero.
Then, check if the agent has any food items in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one food item in the perception object. Loop through the food items within the later and calculate the total number.
Definition at line 8144 of file m_neuro.f90.

8.10.3.158 perception_food_mass_below_calculate()

```
elemental real(srp) function the_neurobio::perception_food_mass_below_calculate (
            class(perception), intent(in) this )
```

Calculate the average mass of a food item from all the items in the current perception object that are **below** the actor agent.

Returns

Average mass of food items within the perception object that are located below (under) the actor agent.

8.10.3.158.1 Implementation details First, initialise the return average mass and the counter for calculating the average both to zero.
Then, check if the agent has any food items in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one food item in the perception object. Calculation of the average mass of the food items **below** is done by concurrent looping through the food items within the perception object.
This is done by checking the condition:

```
if ( food_item .below. this ) ...
```

Note

This uses the user defined operator `.below.` that is implemented in the_environment module.

The average mass of the food items is calculated using the food items mass values returned by the function get_mass (the_environment::food_item::get_mass()).
Final average value is calculated, obviously, by division of the total mass by the total count. In case the count is zero, also return zero mean.
Definition at line 8189 of file m_neuro.f90.

8.10.3.159 perception_food_mass_below_horiz_calculate()

```
elemental real(srp) function the_neurobio::perception_food_mass_below_horiz_calculate (
            class(perception), intent(in) this,
            real(srp), intent(in) hz_lower,
            real(srp), intent(in) hz_upper )
```

Calculate the average mass of a food item from all the items in the current perception object that are **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the this actor agent: [z+hz_lower, z+hz_upper].

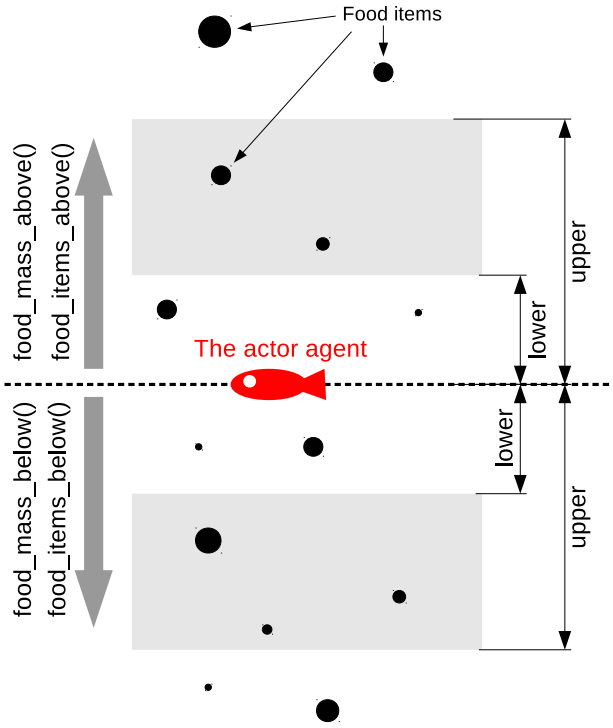


Figure 8.31 Calculation of food items above and below

Parameters

in	*hz_lower*	hz_lower The upper limit for the vertical horizon
in	*hz_upper*	hz_upper The lower limit for the vertical horizon

Returns

Average mass of food items within the perception object that are located below (under) the actor agent.

8.10.3.159.1 Implementation details First, initialise the return average mass and the counter for calculating the average both to zero.
Then, check if the agent has any food items in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one food item in the perception object. Calculation of the average mass of the food items **below** is done by concurrent looping through the food items within the perception object.
The average mass of the food items is calculated using the food items mass values returned by the function `get_mass` (`the_environment::food_item::get_mass()`).
Final average value is calculated, obviously, by division of the total mass by the total count. In case the count is zero, also return zero mean.
Definition at line 8251 of file m_neuro.f90.

8.10.3.160 perception_food_items_above_calculate()

```
elemental integer function the_neurobio::perception_food_items_above_calculate (
            class(perception), intent(in) this )
```

Calculate the number of food items in the perception object that are located **above** the actor agent.

Returns

The number of food items within the perception object that are located above (over) the actor agent.

8.10.3.160.1 Implementation details First, initialise the counter to zero.
Then, check if the agent has any food items in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one food item in the perception object. Calculate food items within the perception that are *above* the agent.
Definition at line 8313 of file m_neuro.f90.

8.10.3.161 perception_food_items_above_horiz_calculate()

```
elemental integer function the_neurobio::perception_food_items_above_horiz_calculate (
            class(perception), intent(in) this,
            real(srp), intent(in) hz_lower,
            real(srp), intent(in) hz_upper )
```

Calculate the number of food items in the perception object that are located **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z-hz_upper, z-hz_upper].

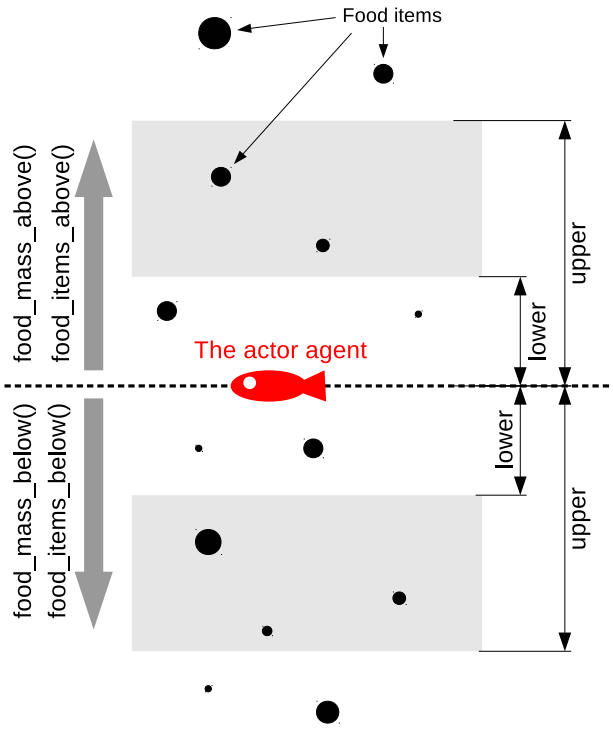


Figure 8.32 Calculation of food items above and below

Parameters

in	*hz_lower*	hz_lower The upper limit for the vertical horizon
in	*hz_upper*	hz_upper The lower limit for the vertical horizon

Returns

The number of food items within the perception object that are located above (over) the actor agent.

8.10.3.161.1 Implementation details First, initialise the counter to zero.
Then, check if the agent has any food items in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one food item in the perception object. Loop through the food items within the later and calculate the total number.
Definition at line 8345 of file m_neuro.f90.

8.10.3.162 perception_food_mass_above_calculate()

```
elemental real(srp) function the_neurobio::perception_food_mass_above_calculate (
            class(perception), intent(in) this )
```

Calculate the average mass of a food item from all the items in the current perception object that are **above** the actor agent.

8.10.3.162.1 Implementation details First, initialise the return average mass and the counter for calculating the average both to zero.
Then, check if the agent has any food items in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one food item in the perception object. Calculation of the average mass of the food items **above** is done by concurrent looping through the food items within the perception object.
This is done by checking the condition:

```
if ( food_item .above. this ) ...
```

Note

This uses the user defined operator `.above.` that is implemented in the_environment module.

The average mass of the food items is calculated using the food items mass values returned by the function get_mass (the_environment::food_item::get_mass()).
Final average value is calculated, obviously, by division of the total mass by the total count. In case the count is zero, also return zero mean.
Definition at line 8390 of file m_neuro.f90.

8.10.3.163 perception_food_mass_above_horiz_calculate()

```
elemental real(srp) function the_neurobio::perception_food_mass_above_horiz_calculate (
            class(perception), intent(in) this,
            real(srp), intent(in) hz_lower,
            real(srp), intent(in) hz_upper )
```

Calculate the average mass of a food item from all the items in the current perception object that are **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z-hz_upper, z-hz_upper].

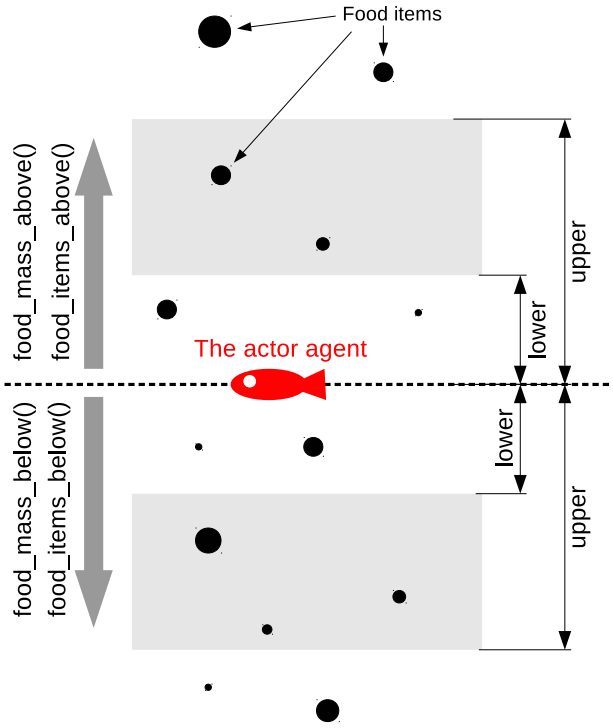


Figure 8.33 Calculation of food items above and below

Parameters

in	*hz_lower*	hz_lower The upper limit for the vertical horizon
in	*hz_upper*	hz_upper The lower limit for the vertical horizon

Returns

Average mass of food items within the perception object that are located above (over) the actor agent.

8.10.3.163.1 Implementation details First, initialise the return average mass and the counter for calculating the average both to zero.
Then, check if the agent has any food items in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one food item in the perception object. Calculation of the average mass of the food items **above** is done by concurrent looping through the food items within the perception object.
The average mass of the food items is calculated using the food items mass values returned by the function `get_mass` (`the_environment::food_item::get_mass()`).
Final average value is calculated, obviously, by division of the total mass by the total count. In case the count is zero, also return zero mean.
Definition at line 8450 of file m_neuro.f90.

8.10.3.164 perception_conspecifics_below_calculate()

```
elemental integer function the_neurobio::perception_conspecifics_below_calculate (
            class(perception), intent(in) this )
```

Calculate the number of conspecifics in the perception object that are located **below** the actor agent.

Returns

The number of conspecifics within the perception object that are located below (under) the actor agent.

8.10.3.164.1 Implementation details First, initialise the counter to zero.
Then, check if the agent has any conspecifics in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one conspecific in the perception object. Loop through the conspecifics and count their total number.
Definition at line 8512 of file m_neuro.f90.

8.10.3.165 perception_conspecifics_above_calculate()

```
elemental integer function the_neurobio::perception_conspecifics_above_calculate (
            class(perception), intent(in) this )
```

Calculate the number of conspecifics in the perception object that are located **above** the actor agent.

Returns

The number of conspecifics within the perception object that are located above (over) the actor agent.

8.10.3.165.1 Implementation details First, initialise the counter to zero.
Then, check if the agent has any conspecifics in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one conspecific in the perception object. Loop through the conspecifics and count their total number.
Definition at line 8539 of file m_neuro.f90.

8.10.3.166 perception_conspecifics_below_horiz_calculate()

```
elemental integer function the_neurobio::perception_conspecifics_below_horiz_calculate (
            class(perception), intent(in) this,
            real(srp), intent(in) hz_lower,
            real(srp), intent(in) hz_upper )
```

Calculate the number of conspecifics in the perception object that are located **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z+hz_lower, z+hz_upper].

Parameters

in	*hz_lower*	hz_lower The upper limit for the vertical horizon
in	*hz_upper*	hz_upper The lower limit for the vertical horizon

Returns

The number of conspecifics within the perception object that are located below (under) the actor agent.

8.10.3.166.1 Implementation details First, initialise the counter to zero.
Then, check if the agent has any conspecifics in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one conspecific in the perception object. Loop through the conspecifics within the later and calculate their number.
Definition at line 8569 of file m_neuro.f90.

8.10.3.167 perception_conspecifics_above_horiz_calculate()

```
elemental integer function the_neurobio::perception_conspecifics_above_horiz_calculate (
            class(perception), intent(in) this,
            real(srp), intent(in) hz_lower,
            real(srp), intent(in) hz_upper )
```

Calculate the number of conspecifics in the perception object that are located **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z-hz_upper, z-hz_upper].

Parameters

in	*hz_lower*	hz_lower The upper limit for the vertical horizon
in	*hz_upper*	hz_upper The lower limit for the vertical horizon

Returns

The number of conspecifics within the perception object that are located above (over) the actor agent.

8.10.3.167.1 Implementation details First, initialise the counter to zero.
Then, check if the agent has any conspecifics in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one conspecific in the perception object. Loop through the conspecifics within and calculate their number.
Definition at line 8613 of file m_neuro.f90.

8.10.3.168 perception_predator_below_calculate()

```
elemental integer function the_neurobio::perception_predator_below_calculate (
            class(perception), intent(in) this )
```

Calculate the number of predators in the perception object that are located **below** the actor agent.

Returns

The number of predators within the perception object that are located below (under) the actor agent.

8.10.3.168.1 Implementation details First, initialise the counter to zero.
Then, check if the agent has any predators in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one conspecific in the perception object. Loop through the predators and count their total number.
Definition at line 8654 of file m_neuro.f90.

8.10.3.169 perception_predator_above_calculate()

```
elemental integer function the_neurobio::perception_predator_above_calculate (
            class(perception), intent(in) this )
```

Calculate the number of predators in the perception object that are located **above** the actor agent.

Returns

The number of predators within the perception object that are located above (over) the actor agent.

8.10.3.169.1 Implementation details First, initialise the counter to zero.
Then, check if the agent has any predators in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one conspecific in the perception object. Loop through the predators and count their total number.
Definition at line 8681 of file m_neuro.f90.

8.10.3.170 perception_predator_below_horiz_calculate()

```
elemental integer function the_neurobio::perception_predator_below_horiz_calculate (
            class(perception), intent(in) this,
            real(srp), intent(in) hz_lower,
            real(srp), intent(in) hz_upper )
```

Calculate the number of predators in the perception object that are located **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z+hz_lower, z+hz_upper].

Parameters

in	*hz_lower*	hz_lower The upper limit for the vertical horizon
in	*hz_upper*	hz_upper The lower limit for the vertical horizon

Returns

The number of predators within the perception object that are located below (under) the actor agent.

8.10.3.170.1 Implementation details First, initialise the counter to zero.
Then, check if the agent has any predators in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one conspecific in the perception object. Loop through the predators within the later and calculate their number.
Definition at line 8711 of file m_neuro.f90.

8.10.3.171 perception_predator_above_horiz_calculate()

```
elemental integer function the_neurobio::perception_predator_above_horiz_calculate (
            class(perception), intent(in) this,
            real(srp), intent(in) hz_lower,
            real(srp), intent(in) hz_upper )
```

Calculate the number of predators in the perception object that are located **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z-hz_upper, z-hz_upper].

Parameters

in	*hz_lower*	hz_lower The upper limit for the vertical horizon
in	*hz_upper*	hz_upper The lower limit for the vertical horizon

Returns

The number of predators within the perception object that are located above (over) the actor agent.

8.10.3.171.1 Implementation details First, initialise the counter to zero.
Then, check if the agent has any predators in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one conspecific in the perception object. Loop through the predators within and calculate their number.
Definition at line 8755 of file m_neuro.f90.

8.10.3.172 perception_food_dist_below_calculate()

```
elemental real(srp) function the_neurobio::perception_food_dist_below_calculate (
            class(perception), intent(in) this )
```

Calculate the average distance to all food items in the current perception object that are **below** the actor agent.

Returns

The average distance to food items within the perception object that are located below (under) the actor agent.

8.10.3.172.1 Implementation details First, initialise the return average and the counter to zero.
Then, check if the agent has any food items in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one food item in the perception object. Calculation of the average distance to the food items **below** is done by concurrent looping through the food items within the perception object and calculating the distance from the agent.
This is done by checking the condition:

```
if ( food_item .below. this ) ...
```

Final average value is calculated, obviously, by division of the total distance by the count. In case the count is zero, also return commondata::missing mean. Note that zero is not returned here because zero distance to food item would result in the highest probability of capture which is not what is intended (zero probability should be invoked for null food items).
Definition at line 8796 of file m_neuro.f90.

8.10.3.173 perception_food_dist_above_calculate()

```
elemental real(srp) function the_neurobio::perception_food_dist_above_calculate (
            class(perception), intent(in) this )
```

Calculate the average distance to all food items in the current perception object that are **above** the actor agent.

Returns

The average distance to food items within the perception object that are located above (over) the actor agent.

8.10.3.173.1 Implementation details First, initialise the return average and the counter to zero.
Then, check if the agent has any food items in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one food item in the perception object. Calculation of the average distance to the food items **above** is done by concurrent looping through the food items within the perception object and calculating the distance from the agent.
This is done by checking the condition:

```
if ( food_item .above. this ) ...
```

Final average value is calculated, obviously, by division of the total distance by the count. In case the count is zero, also return commondata::missing mean. Note that zero is not returned here because zero distance to food item would result in the highest probability of capture which is not what is intended (zero probability should be invoked for null food items).
Definition at line 8852 of file m_neuro.f90.

8.10.3.174 perception_consp_dist_below_calculate()

```
elemental real(srp) function the_neurobio::perception_consp_dist_below_calculate (
            class(perception), intent(in) this )
```

Calculate the average distance to all conspecifics in the current perception object that are **below** the actor agent.

Returns

The average distance to conspecifics within the perception object that are located below (under) the actor agent.

8.10.3.174.1 Implementation details First, initialise the return average and the counter to zero.
Then, check if the agent has any conspecifics in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one conspecific in the perception object. Calculation of the average distance to the conspecifics **below** is done by concurrent looping through the conspecifics within the perception object and calculating the distance from the agent.
This is done by checking the condition:

```
if ( food_item .below. this ) ...
```

Final average value is calculated, obviously, by division of the total distance by the count. In case the count is zero, also return commondata::missing mean.
Definition at line 8908 of file m_neuro.f90.

8.10.3.175 perception_consp_dist_above_calculate()

```
elemental real(srp) function the_neurobio::perception_consp_dist_above_calculate (
            class(perception), intent(in) this )
```

Calculate the average distance to all conspecifics in the current perception object that are **above** the actor agent.

Returns

The average distance to conspecifics within the perception object that are located above (over) the actor agent.

8.10.3.175.1 Implementation details First, initialise the return average and the counter to zero.
Then, check if the agent has any conspecifics in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one conspecific in the perception object. Calculation of the average distance to the conspecifics **above** is done by concurrent looping through the conspecifics within the perception object and calculating the distance from the agent.
This is done by checking the condition:

```
if ( food_item .above. this ) ...
```

Final average value is calculated, obviously, by division of the total distance by the count. In case the count is zero, also return commondata::missing mean.
Definition at line 8961 of file m_neuro.f90.

8.10.3.176 perception_predator_dist_below_calculate()

```
elemental real(srp) function the_neurobio::perception_predator_dist_below_calculate (
            class(perception), intent(in) this )
```

Calculate the average distance to all predators in the current perception object that are **below** the actor agent.

Returns

The average distance to predators within the perception object that are located below (under) the actor agent.

8.10.3.176.1 Implementation details First, initialise the return average and the counter to zero.
Then, check if the agent has any predators in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one conspecific in the perception object. Calculation of the average distance to the predators **below** is done by concurrent looping through the predators within the perception object and calculating the distance from the agent.
This is done by checking the condition:

```
if ( food_item .below. this ) ...
```

Final average value is calculated, obviously, by division of the total distance by the count. In case the count is zero, also return commondata::missing mean.
Definition at line 9014 of file m_neuro.f90.

8.10.3.177 perception_predator_dist_above_calculate()

```
elemental real(srp) function the_neurobio::perception_predator_dist_above_calculate (
            class(perception), intent(in) this )
```

Calculate the average distance to all predators in the current perception object that are **above** the actor agent.

Returns

The average distance to predators within the perception object that are located above (over) the actor agent.

8.10.3.177.1 Implementation details First, initialise the return average and the counter to zero.
Then, check if the agent has any predators in the perception; if not, return zero straight away.
From now on it is assumed that the agent has at least one conspecific in the perception object. Calculation of the average distance to the predators **above** is done by concurrent looping through the predators within the perception object and calculating the distance from the agent.
This is done by checking the condition:

```
if ( food_item .above. this ) ...
```

Final average value is calculated, obviously, by division of the total distance by the count. In case the count is zero, also return commondata::missing mean.
Definition at line 9067 of file m_neuro.f90.

8.10.3.178 predator_capture_probability_calculate_spatobj()

```
real(srp) function the_neurobio::predator_capture_probability_calculate_spatobj (
            class(perception), intent(in) this,
            class(spatialobj_percept_comp), intent(in) this_predator,
            real(srp), intent(in) attack_rate,
            logical, intent(in), optional is_freezing,
            integer, intent(in), optional time_step_model )
```

Calculate the probability of attack and capture of the `this` agent by the predator `this_predator`. This probability is a function of the distance between the predator and the agent and is calculated by the predator-class-bound procedure the_environment::predator::risk_fish(). Example call:

```
risk=proto_parents%individual(ind)%risk_pred(                          &
  proto_parents%individual(ind)%perceive_predator%predators_seen(i),   &
  proto_parents%individual(ind)%perceive_predator%predators_attack_rates(i))
```

Note

Note that this version of the procedure accepts `this_predator` parameter as class the_neurobio::spatialobj_percept_comp that is used for keeping the predator representations in the **perception object**. This representation keeps two separate array for the_neurobio::spatialobj_percept_comp spatial objects and the attack rate.

Parameters

in	*this_predator*	this_predator the predator that is about to attack the agent.

Note

Note that the predator has the SPATIALOBJ_PERCEPT_COMP type that is used in the predator perception object

Parameters

in	*attack_rate*	attack_rate attack rate of the predator.

Note

Note that the predator perception object keeps a separate array of the attack rate.

Parameters

in	*is_freezing*	is_freezing optional logical flag indicating that the fish prey agent is immobile (freezing) that would result in reduced predation risk. Default value is FALSE.
in	*time_step_model*	time_step_model optional time step of the model, if absent, set from the current time step commondata::global_time_step_model_current.

8.10.3.178.1 Checks First, check if the agent has any predators and return zero and exit if there are no predators in the agent's perception object.

Note

This assumes that the predator is much larger than the agent, so the visual range the agent has for detecting the predator is longer than the visual range of the predator for detecting the prey agent.

Warning

The version working with the agent's perception component the_neurobio::predator_capture_probability_calculate_pred() returns a small non-zero probability of capture in contrast to this version accepting `this_predator` object as a class SPATIALOBJ_PERCEPT_COMP. This is because the former normally calculated the objective predation risk whereas this version, subjective risk in the agent's perception. The agent cannot be aware of a predator that is outside of its perception.

Second, Check optional time step parameter. If unset, use global variable `commondata::global_time_step_model_curre`
Third, create a temporary PREDATOR type object using the standard method `make`. The body size and the spatial position are obtained directly from the `this_predator` object. However, the attack rate is obtained from the second dummy argument `attack_rate` to this procedure.

8.10.3.178.2 Implementation Calculate the distance between the agent and predator.
Set the debug plot file name that will be passed to the predator-class-bound function the_environment::predator::risk_fish().
Calculate the probability of capture of the `this` prey agent by the predator. See the_environment::predator::risk_fish() for the details of the calculation.
Definition at line 9134 of file m_neuro.f90.

8.10.3.179 predator_capture_probability_calculate_pred()

```
real(srp) function the_neurobio::predator_capture_probability_calculate_pred (
              class(perception), intent(in) this,
              class(predator), intent(in) this_predator,
              logical, intent(in), optional is_freezing,
              integer, intent(in), optional time_step_model )
```

Calculate the probability of attack and capture of the `this` agent by the predator `this_predator`. This probability is a function of the distance between the predator and the agent and is calculated by the predator-class-bound procedure the_environment::predator::risk_fish().

Note

Note that this version of the procedure accepts `this_predator` parameter as class the_neurobio↩ ::predator, i.e. for the **objective predator object**.

Parameters

in	*this_predator*	this_predator the predator that is about to attack the agent.
in	*is_freezing*	is_freezing optional logical flag indicating that the fish prey agent is immobile (freezing) that would result in reduced predation risk. Default value is FALSE.
in	*time_step_model*	time_step_model optional time step of the model, if absent, set from the current time step commondata::global_time_step_model_current.

8.10.3.179.1 Checks First, check if the agent has any predators in the perception object. Return a near-zero value defined by the commondata::predator_attack_capture_probability_min parameter constant, and exit if there are no predators in the agent's perception object.

Note

This assumes that the predator is much larger than the agent, so the visual range the agent has for detecting the predator is longer than the visual range of the predator for detecting the prey agent.

Warning

The version working with the agent's **perception** component the_neurobio::predator_capture_probability_calculate_spatobj() returns **zero** probability in contrast to this version accepting `this_predator` object as a type PREDATOR. This is because the former normally calculated the subjective assessment of the predation risk whereas this version, objective risk.

Second, Check optional time step parameter. If unset, use global variable `commondata::global_time_step_model_curre`

8.10.3.179.2 Implementation Calculate the distance between the agent and predator.
Set the debug plot file name that will be passed to the predator-class-bound function the_environment::predator::risk_fish().
Calculate the probability of capture of the `this` prey agent by the predator. See the_environment::predator::risk_fish()
for the details of the calculation.
Definition at line 9253 of file m_neuro.f90.

8.10.3.180 predation_capture_probability_risk_wrapper()

```
real(srp) function the_neurobio::predation_capture_probability_risk_wrapper (
             class(perception), intent(in) this,
             logical, intent(in), optional is_freezing )
```

Calculate the overall direct predation risk for the agent, i.e. the probability of attack and capture by the nearest predator.

Parameters

in	*is_freezing*	is_freezing optional logical flag indicating that the fish prey agent is immobile (freezing) that would result in reduced predation risk. Default value is FALSE.

Returns

Returns the probability of capture by the nearest predator.

Definition at line 9344 of file m_neuro.f90.

8.10.3.181 get_prop_size()

```
elemental real(srp) function the_neurobio::get_prop_size (
             class(spatial), intent(in) this )
```

Get the body size property of a polymorphic object. The object can be of the following extension of the basic the_environment::spatial class:

- the_neurobio::conspec_percept_comp - perception object
- the_body::condition - real conspecific.

Note

Other specific classes can be similarly implemented.

Warning

This is not a type-bound function because the base class the_environment::spatial is defined in a different down-level module. Usage: `M = get_props_size(object).`

Returns

the body size of the input the_environment::spatial class object.

8.10.3.181.0.1 Implementation notes Get the properties of the conspecific from the perception object or real physical conspecific data. This is done by determining the `this` data type with "select type" construct.

- if the `this` is a conspecific from the perception object, its body length is obtained from the respective data components of the_neurobio::conspec_percept_comp.
- if the `this` is real conspecific (the_neurobio::condition class), its body length is obtained from the the_body::condition::get_length() and the_body::condition::get_mass() methods.
- in the case construct "default" case, the class is undefined, return commondata::missing value.

Definition at line 9384 of file m_neuro.f90.

8.10.3.182 get_prop_mass()

```
elemental real(srp) function the_neurobio::get_prop_mass (
            class(spatial), intent(in) this )
```

Get the body mass property of a polymorphic object. The object can be of the following extension of the basic the_environment::spatial class:

- the_neurobio::conspec_percept_comp - perception object
- the_body::condition - real conspecific.

Note

Other specific classes can be similarly implemented.

Warning

This is not a type-bound function because the base class the_environment::spatial is defined in a different down-level module. Usage: `M = get_props_mass(object)`.

Returns

the body mass of the input the_environment::spatial class object.

8.10.3.182.0.1 Implementation notes Get the properties of the conspecific from the perception object or real physical conspecific data. This is done by determining the `this` data type with "select type" construct.

- if the `this` is a conspecific from the perception object, its body length is obtained from the respective data components of the_neurobio::conspec_percept_comp.
- if the `this` is real conspecific (the_neurobio::condition class), its body length is obtained from the the_body::condition::get_length() and the_body::condition::get_mass() methods.
- in the case construct "default" case, the class is undefined, return commondata::missing value.

Definition at line 9425 of file m_neuro.f90.

8.10.4 Variable Documentation

8.10.4.1 modname

```
character (len=*), parameter, private the_neurobio::modname = "(THE_NEUROBIO)"  [private]
```

Definition at line 25 of file m_neuro.f90.

8.11 the_population Module Reference

Define the population of agents object, its properties and functions.

Data Types

- type member_population

  Definition of individual member of a population.
- type population

  Definition of the population object.

Functions/Subroutines

- subroutine set_individual_id (this, idnumber)

  Set integer ID number to individual member of the population object.
- integer function get_individual_id (this)

  Get integer ID number to individual member of the population object.
- subroutine individ_posit_in_environ_uniform (this, environ)

  *Places the individual agent, a member of the population, within a specific environment at random with a **uniform** distribution. The agents can be positioned with respect to their initial depth.*
- subroutine genome_individual_set_dead_non_pure (this, non_debug_log)

  *Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function.*
- subroutine init_population_random (this, pop_size, pop_number_here, pop_name_here)

  Initialise the population object.
- subroutine population_destroy_deallocate_objects (this)

  Destroys this population and deallocates the array of individual member objects.
- subroutine population_birth_mortality_init (this, energy_mean, energy_sd)

  Impose selective mortality at birth on the agents. Selective mortality sets a fixed limit on uncontrolled evolution of the energy reserves in newborn agents. If some newborn has too high energy at birth (genetically fixed), such a deviating agent is killed at once.
- integer function population_get_popsize (this)

  Accessor get-function for the size of this population.
- integer function population_get_pop_number (this)

  Accessor get-function for the population number ID.
- character(len=label_length) function population_get_pop_name (this)

  Accessor get-function for the population character label ID.
- subroutine reset_population_id_random (this, pop_number_here, pop_name_here)

  Reset individual IDs of the population members.
- subroutine sex_initialise_from_genome (this)

  Determine the sex for each member of the population.
- subroutine position_individuals_uniform (this, environ)

  Position each member of the population randomly within a bounding environment.
- elemental subroutine sort_population_by_fitness (this)

  This subroutine sorts the population `individual` object by their %fitness components.
- subroutine population_rwalk3d_all_agents_step (this, dist_array, cv_array, dist_all, cv_all, environment_↩ limits, n_walks)

  Perform one or several steps of random walk by all agents.
- subroutine population_rwalk25d_all_agents_step (this, dist_array_xy, cv_array_xy, dist_array_depth, cv_↩ array_depth, dist_all_xy, cv_all_xy, dist_all_depth, cv_all_depth, environment_limits, n_walks)

  Perform one or several steps of random walk by all agents.
- subroutine population_subject_predator_attack (this, this_predator, time_step_model)

Subject the population to an attack by a specific predator. The predator acts on agents in its proximity and takes account of the predation confusion and dilution effects (see the_environment::predator::risk_fish_group()).

- subroutine population_subject_other_risks (this)

  Subject the population to mortality caused by habitat-specific mortality risk. Each agent is affected by the risk associated with the habitat it is currently in.
- subroutine population_subject_individual_risk_mortality (this)

  Subject all members of this population to their individual mortality risks.
- subroutine population_lifecycle_step_preevol (this)

  *This procedure performs a **single step** of the life cycle of the whole population, the agents for the step are selected in a random order.*
- subroutine population_lifecycle_step_eatonly_preevol (this)

  *This procedure performs a **single step** of the life cycle of the whole population, the agents for the step are selected in a random order.*
- subroutine population_save_data_all_agents_csv (this, csv_file_name, save_header, is_logging, is_success)

  Save data for all agents within the population into a CSV file.
- subroutine population_save_data_all_genomes (this, csv_file_name, is_success)

  Save the genome data of all agents in this population to a CSV file.
- subroutine population_load_data_all_genomes (this, pop_size, pop_number_here, pop_name_here, csv_↩ file_name, missing_random, is_success)

  Load the genome data of all agents in this population from a CSV file. Note that the procedure implements several error correcting measures, e.g. checks for minimum number of rows in the file and minimum row length. The input CSV file therefore can include short text notes that are then ignored when reading data.
- subroutine population_save_data_memory (this, csv_file_name, is_success)

  Save the perceptual and emotional memory stack data of all agents in this population to a CSV file.
- subroutine population_save_data_movements (this, csv_file_name, is_success)

  Save the latest movement history of all agents. This method makes use of the the_environment::spatial_moving::history structure that saves latest movements of each agent.
- subroutine population_save_data_behaviours (this, csv_file_name, is_success)

  Save the behaviours history stack the_neurobio::behaviour::history_behave for all agents.
- pure subroutine population_preevol_fitness_calc (this)

  *Calculate fitness for the pre-evolution phase of the genetic algorithm. **Pre-evolution** is based on selection for a simple criterion without explicit reproduction etc. The criterion for selection at this phase is set by the integer the_individual::individual_agent::fitness component. This procedure provides a whole-population wrapper for the the_individual::individual_agent::fitness_calc() function.*
- pure integer function population_ga_reproduce_max (this)

  Determine the number of parents that have fitness higher than the minimum acceptable value. Also, only alive agents are included into the reproducing number.
- real(srp) function population_ga_mutation_rate_adaptive (this, baseline, maxvalue)

  This function implements adaptive mutation rate that increases as the population size reduces.

Variables

- character(len= ∗), parameter, private modname = "(THE_POPULATION)"
- integer, public global_ind_n_eaten_by_predators

  Global indicator variable that keeps the number of agents that have died as a consequence of predatory attacks. All other dies are therefore caused by starvation.

8.11.1 Detailed Description

Define the population of agents object, its properties and functions.

8.11.2 THE_POPULATION module

The population is in the simplest case an array of individual agent objects. Individual properties of the population members can be referred as, e.g. `proto_parentsindividual(i)fitness`.

8.11.3 Function/Subroutine Documentation

8.11.3.1 set_individual_id()

```
subroutine the_population::set_individual_id (
             class(member_population), intent(inout) this,
             integer, intent(in) idnumber )
```

Set integer ID number to individual member of the population object.

Note

Note that this subroutine is private as setting individual IDs makes sense only during the initialisation phase of the population.

Parameters

in,out	*this*	class, member of population.
in	*idnumber*	id, integer id number assigned.

person_number is assigned the idnumber
Definition at line 191 of file m_popul.f90.

8.11.3.2 get_individual_id()

```
integer function the_population::get_individual_id (
             class(member_population), intent(in) this )
```

Get integer ID number to individual member of the population object.

Note

Note that this is public, so we can retrieve individual IDs but not set them (id's are always set at initialisation)

Parameters

in	*this*	class, member of population.

Returns

Parameters

id,integer	id number retreived.

person_number is assigned the idnumber
Definition at line 205 of file m_popul.f90.

8.11.3.3 individ_posit_in_environ_uniform()

```
subroutine the_population::individ_posit_in_environ_uniform (
             class(member_population), intent(inout) this,
             class(environment), intent(in) environ )
```

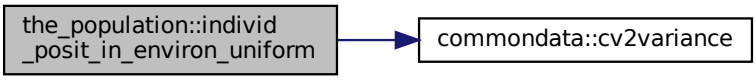
Places the individual agent, a member of the population, within a specific environment at random with a **uniform** distribution. The agents can be positioned with respect to their initial depth.

- fixed depth, see commondata::init_agents_depth_is_fixed
- Gaussian depth, see commondata::init_agents_depth_is_gauss
- fully uniform distribution.

Parameters

in	*environ*	environ sets the environment object where the agent is placed randomly

Definition at line 223 of file m_popul.f90.
Here is the call graph for this function:



8.11.3.4 genome_individual_set_dead_non_pure()

```
subroutine the_population::genome_individual_set_dead_non_pure (
            class(member_population), intent(inout) this,
            logical, intent(in), optional non_debug_log )
```

Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function.

Note

This is a non-pure function logging some extended diagnostics about the dying agent. See `dies` procedure from the_genome and all its overrides:

- the_genome::individual_genome::dies();
- the_neurobio::appraisal::dies();
- the_neurobio::gos_global::dies();
- the_individual::individual_agent::dies().

Warning

This function should be normally used only while **debugging**. In normal runs use the pure subroutine `dies` from `ÌNDIVIDUAL_GENOME`.

Local flag to do show log.
Local parameter showing how many latest memory values to log out.
Don't show log by default.
This is a pure function from `THE_GENOME` level.
Turn on output logging only if in the DEBUG mode or if `non_debug_log` is explicitly set to TRUE.
Just exit back if no logging.

Note

Note that `TOSTR` accepts arrays including (concatenated) character arrays.

Definition at line 261 of file m_popul.f90.

8.11.3.5 init_population_random()

```
subroutine the_population::init_population_random (
            class(population), intent(inout) this,
            integer, intent(in) pop_size,
            integer, intent(in), optional pop_number_here,
            character (len=*), intent(in), optional pop_name_here )
```

Initialise the population object.
Initialise the population object, init it with random individuals (function init on individuals), and assign sequential person_number`s.

Parameters

in,out	*this*	class, This – member of population class (this)).
in	*pop_size*	pop_size, The size of the population.
in	*pop_number_here*	id, Optional numeric population ID.
in	*pop_name_here*	descriptor, Optional population string descriptor.

Set population size from input parameter.
Allocate the population
Initialise all individuals of the population
first, call `init to object individual(i)
Set optional descriptors for the whole population. Numeric ID and a short text description.
If optional ID is absent, ID is set to a random integer value from 1 to the maximum integer allowed for the pop_↩ number type (minus 1).
If optional text description string of the population is absent, it is set to the string representation of its numeric ID.
Log the initial location of the agents.

Warning

The logic of the logger constructs here must coincide with that in the population::individ_posit_in_environ_uniform().

Definition at line 318 of file m_popul.f90.

8.11.3.6 population_destroy_deallocate_objects()

```
subroutine the_population::population_destroy_deallocate_objects (
            class(population), intent(inout) this )
```

Destroys this population and deallocates the array of individual member objects.
Definition at line 391 of file m_popul.f90.

8.11.3.7 population_birth_mortality_init()

```
subroutine the_population::population_birth_mortality_init (
            class(population), intent(inout) this,
            real(srp), intent(in), optional energy_mean,
            real(srp), intent(in), optional energy_sd )
```

Impose selective mortality at birth on the agents. Selective mortality sets a fixed limit on uncontrolled evolution of the *energy reserves* in newborn agents. If some newborn has too high energy at birth (genetically fixed), such a deviating agent is killed at once.
The values of the risk are chosen such that it is zero at the fixed population mean (agent does not deviate) and reaches 1.0 if the agent's energy reserves are 3.0 standard deviations higher than the fixed mean value (agent strongly deviates).

Parameters

in	*energy_mean*	energy_mean optional mean energy at birth, if absent, is calculated from the population data.

Parameters

in	*energy_sd*	energy_sd optional mean energy at birth, if absent, is calculated from the population data.

- MORTALITY_BIRTH_INIT_ENERG_ABSCISSA is the baseline abscissa for the nonparametric interpolation function grid, in units of the standard deviation (sigma) of the energy reserves in the newborn population, i.e. the_body::condition::energy_birth.
- MORTALITY_BIRTH_INIT_ENERG_ORDINATE is the ordinate for the nonparametric interpolation function grid: sets the probability of the agent's death given its energy reserves deviate from the fixed mean by the number of standard deviations set by MORTALITY_BIRTH_INIT_ENERG_ABSCISSA.

Selective mortality sets a fixed limit on uncontrolled evolution of the energy reserves in newborn agents. All newborn agents that have the energy reserves exceeding some point may die with the probability determined by the grid arrays:

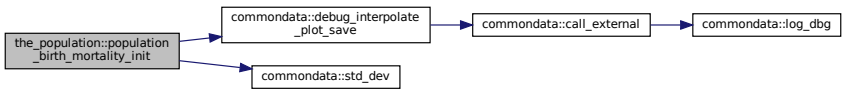
- MORTALITY_BIRTH_INIT_ENERG_ABSCISSA;
- MORTALITY_BIRTH_INIT_ENERG_ORDINATE.

The probability of death (mortality risk) is determined by the nonparametric nonlinear function defined by DDPINTERPOL, where the actual grid abscissa is the energy reserve of the agent in and grid ordinate is the mortality risk.
The values of the risk are chosen such that it is zero at the fixed population mean (agent does not deviate) and reaches 1.0 if the agent's energy reserves are 3.0 standard deviations higher than the mean value.
Interpolation plots can be saved in the debug mode using this plotting command: commondata::debug_interpolate_plot_

Warning

Involves **huge** number of plots, should normally be disabled.

Definition at line 411 of file m_popul.f90.
Here is the call graph for this function:



8.11.3.8 population_get_popsize()

```
integer function the_population::population_get_popsize (
            class(population), intent(in) this )
```

Accessor get-function for the size of this population.
Definition at line 510 of file m_popul.f90.

8.11.3.9 population_get_pop_number()

```
integer function the_population::population_get_pop_number (
            class(population), intent(in) this )
```

Accessor get-function for the population number ID.
Definition at line 520 of file m_popul.f90.

8.11.3.10 population_get_pop_name()

```
character(len=label_length) function the_population::population_get_pop_name (
            class(population), intent(in) this )
```

Accessor get-function for the population character label ID.
Definition at line 530 of file m_popul.f90.

8.11.3.11 reset_population_id_random()

```
subroutine the_population::reset_population_id_random (
            class(population), intent(inout) this,
            integer, intent(in), optional pop_number_here,
            character (len=*), intent(in), optional pop_name_here )
```

Reset individual IDs of the population members.
Makes new random individual IDs for the population members.

Parameters

in,out	*this*	class, This – member of population class (this)).
in	*pop_number_here*	id, Optional numeric population ID.
in	*pop_name_here*	descriptor, Optional population string descriptor.

Reset all individual IDs of the population members
Set optional descriptors for the whole population. Numeric ID and a short text description.
If optional ID is absent, ID is set to a random integer value from 1 to the maximum integer allowed for the pop_↩ number type (minus 1).
If optional text description string of the population is absent, it is set to the string representation of its numeric ID.
Definition at line 541 of file m_popul.f90.

8.11.3.12 sex_initialise_from_genome()

```
subroutine the_population::sex_initialise_from_genome (
            class(population), intent(inout) this )
```

Determine the sex for each member of the population.
Definition at line 579 of file m_popul.f90.

8.11.3.13 position_individuals_uniform()

```
subroutine the_population::position_individuals_uniform (
            class(population), intent(inout) this,
            class(environment), intent(in), optional environ )
```

Position each member of the population randomly within a bounding environment.

Note

Moved the positioning procedure into a separate procedure as initialising population may involve different spatial positioning, e.g. uniform within the bounding environment or gaussian localised.

Parameters

in	*environ*	environ the environment where we place the population

Warning

Even though this parameter is optional, the bounding environment should in most cases (almost?) be fixed, and provided as a parameter. In most cases, unlimited environment is useful for debugging only. TODO: convert to class

Local object representing the bounding environment for this population
Local counter
Check if the bounding environment is provided, if not, place agents without limits

Warning

The bounding environment should in most cases be fixed, and provided as a parameter.

Position agents randomly (uniform distribution) within the bounding environment.
Definition at line 596 of file m_popul.f90.

8.11.3.14 sort_population_by_fitness()

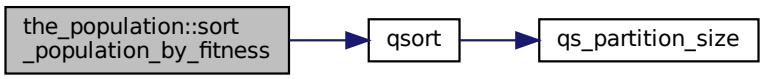
```
elemental subroutine the_population::sort_population_by_fitness (
            class(population), intent(inout) this )
```

This subroutine sorts the population `individual` object by their %fitness components.
The two subroutines `qsort` and `qs_partition_fitness` are a variant of the recursive quick sort algorithm adapted for `MEMBER_POPULATION` integer fitness component

Parameters

`in,out`	*this*	class, This – member of population class (this)).

This is the array component we sort.
Definition at line 637 of file m_popul.f90.
Here is the call graph for this function:



8.11.3.15 population_rwalk3d_all_agents_step()

```
subroutine the_population::population_rwalk3d_all_agents_step (
            class(population), intent(inout) this,
            real(srp), dimension(:), intent(in), optional dist_array,
            real(srp), dimension(:), intent(in), optional cv_array,
            real(srp), intent(in), optional dist_all,
            real(srp), intent(in), optional cv_all,
            class(environment), intent(in), optional environment_limits,
            integer, intent(in), optional n_walks )
```

Perform one or several steps of random walk by all agents.

Note

This procedure was used for debugging.

Parameters

in	*dist_array*	step_size_array an array of step sizes for each individual.
in	*cv_array*	cv_array Coefficients of variation for the walk.
in	*dist_all*	dist_all the value of the walk step size that is identical in all agents within the population.
in	*cv_all*	cv_all the value of the walk coefficient of variation that is identical in all agents within the population.
in	*environment_limits*	environment_limits Limits of the environment area available for the random walk. The moving object cannot get beyond this limit. If this parameter is not provided, the environmental limits are obtained automatically from the global array the_environment::global_habitats_available.
in	*n_walks*	n_walk optional number of walk steps that should be performed, default just one.

8.11.3.15.1 Implementation details

- Calculate the distance array size.

calculate the step size along the axes from the distance array.

- Calculate the random permutation of individual indices.

  Warning

  Random order here is a prototype for testing for use in behaviour selection by population members.

- Perform Gaussian random walks for each of the individuals in a random order that is set by the `pop_↩permutation` array.

Definition at line 725 of file m_popul.f90.

8.11.3.16 population_rwalk25d_all_agents_step()

```
subroutine the_population::population_rwalk25d_all_agents_step (
            class(population), intent(inout) this,
            real(srp), dimension(:), intent(in), optional dist_array_xy,
            real(srp), dimension(:), intent(in), optional cv_array_xy,
            real(srp), dimension(:), intent(in), optional dist_array_depth,
            real(srp), dimension(:), intent(in), optional cv_array_depth,
            real(srp), intent(in), optional dist_all_xy,
            real(srp), intent(in), optional cv_all_xy,
            real(srp), intent(in), optional dist_all_depth,
            real(srp), intent(in), optional cv_all_depth,
            class(environment), intent(in), optional environment_limits,
            integer, intent(in), optional n_walks )
```

Perform one or several steps of random walk by all agents.

Note

This procedure was used for debugging.

Parameters

in	*dist_array_xy*	dist_array_xy an array of step sizes for each individual.
in	*cv_array_xy*	cv_array_xy Coefficients of variation for the walk.
in	*dist_array_depth*	dist_array_depth an array of step sizes for each individual.
in	*cv_array_depth*	cv_array_depth Coefficients of variation for the walk.

Parameters

in	*dist_all_xy*	dist_all_xy the value of the walk step size for horizontal plane that is identical in all agents within the population.
in	*cv_all_xy*	cv_all_xy the value of the walk coefficient of variation in the horizontal plane that is identical in all agents within the population.
in	*dist_all_depth*	dist_all_depth the value of the walk step size for the depth plane that is identical in all agents within the population.
in	*cv_all_depth*	cv_all_depth the value of the walk coefficient of variation in the depth plane that is identical in all agents within the population.
in	*environment_limits*	environment_limits Limits of the environment area available for the random walk. The moving object cannot get beyond this limit. If this parameter is not provided, the environmental limits are obtained automatically from the global array the_environment::global_habitats_available.
in	*n_walks*	n_walk optional number of walk steps that should be performed, default just one.

8.11.3.16.1 Implementation details

- Calculate the distance array size.

If the depth walk step distance is not provided as a parameter, 1/2 of the agent body size is used as the default value. Thus, it is assumed that the extent of random movements of the agents in the horizontal plane is greater than vertical movements.

- Calculate the step size along the axes from the distance array.
- Calculate the random permutation of individual indices.

  Warning

  Random order here is a prototype for testing for use in behaviour selection by population members.

- Perform Gaussian random walks for each of the individuals in a random order that is set by the `pop_↩ permutation` array.

Definition at line 838 of file m_popul.f90.

8.11.3.17 population_subject_predator_attack()

```
subroutine the_population::population_subject_predator_attack (
             class(population), intent(inout) this,
             class(predator), intent(in) this_predator,
             integer, intent(in), optional time_step_model )
```

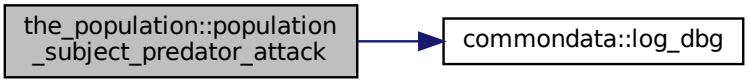
Subject the population to an attack by a specific predator. The predator acts on agents in its proximity and takes account of the predation confusion and dilution effects (see the_environment::predator::risk_fish_group()).

8.11.3.17.1 Notable variables

- p_risk is an array with the size equal to the agent population size, that keeps all the attack probabilities calculated by the the_environment::predator::risk_fish_group() function.

prey_index is the partial index of the prey agents that are in proximity of this predator.

8.11.3.17.2 Implementation notes **Preparations:** Check optional time step parameter. If not provided, use global commondata::global_time_step_model_current parameter value.
First, calculate the risk of predation from this specific predator to each of the agents in the population using the the_environment::predator::risk_fish_group() method. The "raw" indexed output scheme is used here to avoid multiple cycling over the whole large population of agents, only the agents that are closest to the predator are processed and attacked (maximum number is equal to the index limit commondata::predator_risk_group_select_index_partial).
Second, cycle through all the agents in close proximity of the predator, up to the maximum size of partial indexing parameter commondata::predator_risk_group_select_index_partial. The predator then stochastically attacks each of these agents with the probability equal to the risk of predation. If the attack is successful, the agent the_genome::individual_genome::dies() and loop is exited because the predator is assumed to catch only one agent at a time.
Definition at line 1008 of file m_popul.f90.
Here is the call graph for this function:



8.11.3.18 population_subject_other_risks()

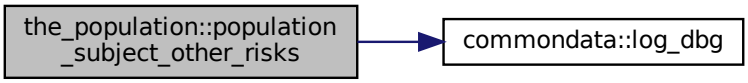
```
subroutine the_population::population_subject_other_risks (
            class(population), intent(inout) this )
```

Subject the population to mortality caused by habitat-specific mortality risk. Each agent is affected by the risk associated with the habitat it is currently in.

Note

Note that there is no such a function for a single agent as it does not seem to be necessary.

8.11.3.18.1 Implementation notes All agents in the population are randomly subjected to the mortality risk the_environment::habitat::risk_mortality that is linked to the habitat object the agent is currently in in a loop.
If the agent is unhappy and is subjected to mortality event, it immediately the_genome::individual_genome::dies().
Definition at line 1102 of file m_popul.f90.
Here is the call graph for this function:



8.11.3.19 population_subject_individual_risk_mortality()

```
subroutine the_population::population_subject_individual_risk_mortality (
            class(population), intent(inout) this )
```

Subject all members of this population to their individual mortality risks.

8.11.3.19.1 Implementation notes The procedure is simple, loop over all individual agents in the population and stochastically call the_genome::individual_genome::dies() method with probability equal to individual mortality of the agent.
Definition at line 1138 of file m_popul.f90.

8.11.3.20 population_lifecycle_step_preevol()

```
subroutine the_population::population_lifecycle_step_preevol (
            class(population), intent(inout) this )
```

This procedure performs a **single step** of the life cycle of the whole population, the agents for the step are selected in a random order.

8.11.3.20.1 Notable variables

- `inds_order` is an array that sets the order in which the agents are being drawn out of the population to perform the step.

`ind_seq` is the sequential number of the agent as drawn from the population; it is the loop control counter variable.

- `ind_real` is the real sequential number of the agent in the population; this real id is obtained from the order array `inds_order`.

8.11.3.20.2 Implementation notes First, an ordering array `inds_order` is calculated that sets the order in which the agents are drawn from the population. In the simplest case, the order of the agents is random, so this array is actually an array of random integers. The procedure `PERMUTE_RANDOM` from HEDTOOLS is used here then.
The agents can also be processed in any **non-random** order. This would require invoking an array indexing procedure `ARRAY_INDEX` instead of `PERMUTE_RANDOM`.
For example, to process the agents in the order of the body mass, the ordering array can be obtained obtained as below:

```
call ARRAY_INDEX(this%individual%get_mass(), inds_order)
```

But this code would rank the agents in an *increasing* order of their body mass. If this is not what is expected, e.g. if the non-random selection is used to mimic a competitive advantage for bigger and heavier agents, a reverse of the ordering is obtained like this:

```
inds_order = inds_order(this%population_size:1:-1)
```

Second, the agents are drawn from the population, one by one, in the named loop construct `INDIVIDUALS`.

- One individual is then drawn from the `inds_order` ordering array.

8.11.3.20.2.1 Initialisations

- First, a check is done if the agent is dead or starved to death; if yes, no further processing is done. Also, in the former case the individual_genome::dies() method is called.

`agent_in` index calculates the habitat number where the selected agent is currently in, calling the the_environment::spatial::find_environment() method.

8.11.3.20.2.2 Get perceptions

- Inner perceptions are obtained from the agent's "organism": stomach contents, bodymass, energy, age, reproductive factor: the_neurobio::perception::perceptions_inner().

Simple environmental perceptions are obtained: light, depth: the_neurobio::perception::perceptions_environ().

- Spatial perceptions are obtained for food items, conspecifics and predators in proximity of this agent:
  - the_neurobio::perception::see_food()
  - the_neurobio::perception::see_consp()
  - the_neurobio::perception::see_pred()
- The above perceptions are added into the memory stack calling the_neurobio::perception::perception_to_memory().

8.11.3.20.2.3 Appraisal: Produce motivations

- Perception components are calculated for each of the motivational states of the agent via the neuronal response function(s): the_neurobio::appraisal::motivations_percept_components().

Then, primary motivation values are calculated from the perception components: the_neurobio::appraisal::motivations_primary_calc().

- The values of the primary motivations are subjected to modulation the_neurobio::appraisal::modulation().
- Final motivations of the agent are saved into the emotional memory stack: the_neurobio::appraisal::motivations_to_memory().

8.11.3.20.2.4 Determine the Global Organismic State

- The Global Organismic State of the agent is calculated using the the_neurobio::gos_global::gos_find() method.

The population-wise maximum motivation value that is used for rescaling is calculated now.

8.11.3.20.2.5 Determine and execute the optimal behaviour

- Once the GOS is determined for this agent, it selects the individually optimal behaviour that minimises its expected arousal; this behaviour is then executed. Both steps are implemented in the the_neurobio↩ ::behaviour::do_behave() method.

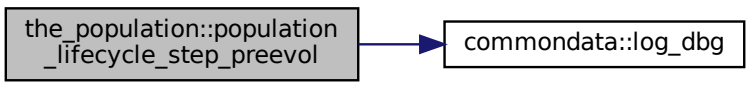
8.11.3.20.2.6 Update characteristics of the agent

- Finally, characteristics of this agent are updated depending on the consequences of the behaviour. For example, hormone levels are updated the_body::condition::sex_steroids_update() and the living cost of the agent is subtracted (the_body::condition::subtract_living_cost()). Digestion also occurs by emptying the stomach by a fixed fraction (the_body::condition::stomach_empify(). Also, the energy reserves of the agent are updated based on the current mass and length (the_body::condition::energy_update(). Note that the agent characteristics that directly follow from the behaviour unit that has been executed (e.g. food gain if the agent decided to eat a food item or travel cost if the agent migrated) are processed and updated in the respective behaviour execution method.

Finally, the age of the agent is incremented to one time step.

- Finally, another check is done if the agent is starved to death, if yes, the agent individual_genome::dies().

Definition at line 1158 of file m_popul.f90.
Here is the call graph for this function:



8.11.3.21 population_lifecycle_step_eatonly_preevol()

```
subroutine the_population::population_lifecycle_step_eatonly_preevol (
            class(population), intent(inout) this )
```

This procedure performs a **single step** of the life cycle of the whole population, the agents for the step are selected in a random order.

Warning

This version of the life cycle step includes only optimal food selection and eating and does not include the full fledged behaviour selection cascade of procedures ::do_behave().

8.11.3.21.1 Notable variables

- `inds_order` is an array that sets the order in which the agents are being drawn out of the population to perform the step.

`ind_seq` is the sequential number of the agent as drawn from the population; it is the loop control counter variable.

- `ind_real` is the real sequential number of the agent in the population; this real id is obtained from the order array `inds_order`.

8.11.3.21.2 Implementation notes First, an ordering array `inds_order` is calculated that sets the order in which the agents are drawn from the population. In the simplest case, the order of the agents is random, so this array is actually an array of random integers. The procedure `PERMUTE_RANDOM` from HEDTOOLS is used here then.
The agents can also be processed in any **non-random** order. This would require invoking an array indexing procedure `ARRAY_INDEX` instead of `PERMUTE_RANDOM`.
For example, to process the agents in the order of the body mass, the ordering array can be obtained obtained as below:

```
call ARRAY_INDEX(this%individual%get_mass(), inds_order)
```

But this code would rank the agents in an ∗increasing ∗order of their of their body mass. If this is not what is expected, e.g. if the non-random selection is used to mimic a competitive advantage for bigger and heavier agents, a reverse of the ordering is obtained like this:

```
inds_order = inds_order(this%population_size:1:-1)
```

Second, the agents are drawn from the population, one by one, in the named loop construct `INDIVIDUALS`.

- One individual is then drawn from the `inds_order` ordering array.

8.11.3.21.2.1 Initialisations

- First, a check is done if the agent is dead or starved to death; if yes, no further processing is done. Also, in the former case the individual_genome::dies() method is called.

`agent_in` index calculates the habitat number where the selected agent is currently in, calling the the_environment::spatial::find_environment() method.

8.11.3.21.2.2 Get perceptions

- Inner perceptions are obtained from the agent's "organism": stomach contents, bodymass, energy, age, reproductive factor: the_neurobio::perception::perceptions_inner().

Simple environmental perceptions are obtained: light, depth: the_neurobio::perception::perceptions_environ().

- Spatial perceptions are obtained for food items, conspecifics and predators in proximity of this agent:
  - the_neurobio::perception::see_food()
  - the_neurobio::perception::see_consp()
  - the_neurobio::perception::see_pred()
- The above perceptions are added into the memory stack calling the_neurobio::perception::perception_to_memory().

8.11.3.21.2.3 Appraisal: Produce motivations

- Perception components are calculated for each of the motivational states of the agent via the neuronal response function(s): the_neurobio::appraisal::motivations_percept_components().

Then, primary motivation values are calculated from the perception components: the_neurobio::appraisal::motivations_primary_calc().

- The values of the primary motivations are subjected to modulation the_neurobio::appraisal::modulation().
- Final motivations of the agent are saved into the emotional memory stack: the_neurobio::appraisal::motivations_to_memory().

8.11.3.21.2.4 Determine the Global Organismic State

- The Global Organismic State of the agent is calculated using the the_neurobio::gos_global::gos_find() method.

The population-wise maximum motivation value that is used for rescaling is calculated now.

8.11.3.21.2.5 Determine and execute the optimal behaviour

- Once the GOS is determined for this agent, it selects the optimal food item that minimises its expected arousal and then tries to eat this food item. However, if the agent has no food in its perception a default random walk is executed.

8.11.3.21.2.6 Update characteristics of the agent

- Finally, characteristics of this agent are updated depending on the consequences of the behaviour. For example, hormone levels are updated and the living cost of the agent is subtracted. Note that the agent characteristics that directly follow from the behaviour unit that has been executed (e.g. food gain if the agent decided to eat a food item or travel cost if the agent migrated) are processed and updated in the respective behaviour execution method.

Finally, the age of the agent is incremented to one time step.

- Finally, another check is done if the agent is starved to death, if yes, the agent individual_genome::dies().

Definition at line 1356 of file m_popul.f90.

8.11.3.22 population_save_data_all_agents_csv()

```
subroutine the_population::population_save_data_all_agents_csv (
             class(population), intent(in) this,
             character(len=*), intent(in) csv_file_name,
             logical, intent(in), optional save_header,
             logical, intent(in), optional is_logging,
             logical, intent(out), optional is_success )
```

Save data for all agents within the population into a CSV file.

Note

Note that this procedure is not using the file_io wrappers.

Parameters

in	*csv_file_name*	csv_file_name is the name of the CSV output file.
in	*save_header*	save_header turn ON/OFF of the descriptive file header. Header is saved into the first row of the CSV output file If not present, default is FALSE.
in	*is_logging*	is_logging turn ON/OFF writing the file name and data into the logger. If not present, default is TRUE if it is the debug mode.
out	*is_success*	is_success Flag showing that data save was successful (if TRUE).

8.11.3.22.1 Implementation notes

8.11.3.22.1.1 Local variables for CSV backend

- handle_csv is the CSV file handle object defining the file name, Fortran unit and error descriptor, see HEDTOOLS manual for details.

csv_record_tmp is the temporary character string that keeps the whole record of the file, i.e. the whole row of the spreadsheet table.

- COLUMNS is a parameter array that keeps all column headers; its size is equal to the total number of variables (columns) in the data spreadsheet file.

8.11.3.22.1.2 Save data in CSV file

Note

Note that this subroutine does not use the object oriented wrappers from the file_io module.

- Define the file name %name component of the CSV file handle. The file handle object handle_csv is now used as the file identifier.

Open the output file defined by the handle_csv handle object for writing.

- Possible error status of the latest file operation is obtained by the %status component of the file handle. Check if there were any errors opening the file and report in the logger with the error tag.
- If the save_header flag is set to TRUE, save the CSV file header.
- Prepare the character string variable csv_record_tmp that keeps the whole record (row) of data in the output CSV data file. The length of this string should be enough to fit all the record data, otherwise the record is truncated.
- Produce the first record containing the column headers (variable names). Note that CSV_RECORD_APPEND() accepts both arrays and scalar values for appending. Also, write the first record physically to the file.
- The actual data are written to the CSV file in a loop over all the individual members of the population. One record (row) of the data file then represents a single individual.
- the csv_record_tmp character string variable is produced such that it can fit the whole record;
- the actual data for the individual is appended to the current record one by one. Note that logical values are converted to integers using commondata::conv_l2r() function.
- after all data are appended to the record, this record is physically written to the disk using CSV_RECORD_WRITE().
- When all the records are saved, the CSV file is closed with CSV_CLOSE().
- This is finally sent to the logger (if logging_enabled is TRUE).

The CSV output data file can be optionally compressed with the commondata::cmd_zip_output command if commondata::is_zip_outputs is set to TRUE.
Definition at line 1554 of file m_popul.f90.

8.11.3.23 population_save_data_all_genomes()

```
subroutine the_population::population_save_data_all_genomes (
             class(population), intent(in) this,
             character(len=*), intent(in) csv_file_name,
             logical, intent(out), optional is_success )
```

Save the genome data of all agents in this population to a CSV file.

Parameters

in	*csv_file_name*	csv_file_name is the name of the CSV output file.
out	*is_success*	is_success Flag showing that data save was successful (if TRUE).

8.11.3.23.1 Implementation notes

8.11.3.23.1.1 Build column names First, determine the number of columns in the data file. The number of columns is calculated by unwinding the whole data structure

- `chromosome(j,k)allele(l)allele_value(m)`

See the_genome for more details on the data structure.
The array of the column names is then allocated to the above number.
The column names are built again by unwinding the whole genome data structure into a linear sequence. The column names are like this:
`CRO_1_1_ALE_01_AC_1, CRO_1_1_ALE_01_AC_2, CRO_1_1_ALE_01_AC_3, ...`

8.11.3.23.1.2 Write data to the file First, the file is opened for writing.
The first record of the data that contains the column names is then "appended" into the complete record and written to the file. The length of this record is calculated based on the length of the columns and their number.

- The remaining columns contain the chromosome and gene labels (see above);

This first line consisting of column names is then written to the output file.
The maximum length of the data record is calculated as the maximum string length of a single data value multiplied by the number of columns. Because the record also adds separators, the number of columns multiplied by three is added to this value.
Finally, cycle over all individuals in this population and save the genome data. The first two columns are the_genome::individual_genome::person_number and the_genome::individual_genome::genome_label. The other columns "unwind" the genome data structure over the inner loops for chromosomes, homologues, alleles and allele components.

- - `chromosome(j,k)allele(l)allele_value(m)`

See the_genome for more details on the data structure.
Once all individuals are saved, the file is closed.
The CSV output data file can be optionally compressed with the commondata::cmd_zip_output command if commondata::is_zip_outputs is set to TRUE.
Definition at line 1841 of file m_popul.f90.

8.11.3.24 population_load_data_all_genomes()

```
subroutine the_population::population_load_data_all_genomes (
             class(population), intent(inout) this,
             integer, intent(in) pop_size,
             integer, intent(in), optional pop_number_here,
             character (len=*), intent(in), optional pop_name_here,
             character(len=*), intent(in) csv_file_name,
             logical, intent(in), optional missing_random,
             logical, intent(out), optional is_success )
```

Load the genome data of all agents in this population from a CSV file. Note that the procedure implements several error correcting measures, e.g. checks for minimum number of rows in the file and minimum row length. The input CSV file therefore can include short text notes that are then ignored when reading data.

Parameters

in	*pop_size*	pop_size, The size of the population.
in	*pop_number_here*	id, Optional numeric population ID.
in	*pop_name_here*	descriptor, Optional population string descriptor.
in	*csv_file_name*	csv_file_name is the name of the CSV output file.
in	*missing_random*	missing_random Flag dictating to generate random genomes if the commondata::popsize parameter exceeds the population size in the CSV genome data size. TRUE = generate random genome, FALSE = leave empty data commondata::unknown

Parameters

out	*is_success*	is_success Flag showing that data save was successful (if TRUE).

- handle_csv is the CSV file handle object defining the file name, Fortran unit and error descriptor, see HEDTOOLS manual for details.
- line_data_buff is the input data buffer that is being read via the CSV_IO procedure READLINE()
- n_rows_in is the number of rows in the input CSV file excluding the first row containing the variable names.
- N_ROWS_MIN is the minimum number of readable rows in the CSV genome data file, if there are less in the file, data are considered invalid.
- MIN_FILE_INP_LEN is the minimum length of the data row to be included in the read genome.
- line_data_substrings - an array of row string data values after parding the whole input raw data.
- matrix_row – array representing the genome data for a single row after parsing the input data file row.

8.11.3.24.1 Implementation notes Genome data are obtained from the CSV data file provided by the intent(in) parameter csv_file_name.

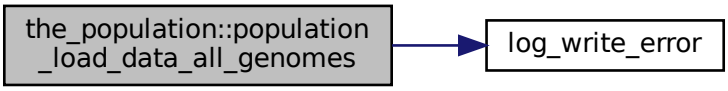
- First, calculate the number of records in the input CSV file and check that this number is greater than the minimum value set by the local parameter N_ROWS_MIN. Also check if the file can be read at all. If either of these conditions hold, exit from the procedure with
- Second, open the input data file for reading
  - Define the file name %name component of the CSV file handle. The file handle object handle_csv is now used as the file identifier.
- Open the output file defined by the handle_csv handle object for reading.
- Possible error status of the latest file operation is obtained by the %status component of the file handle. Check if there were any errors opening the file and report in the logger with the error tag.
- Third, read the variable names line of the CSV file. Note that they are not used here.
- **HAVE_POP** check if the population is allocated and the population size is set, if not, allocate and set to commondata::popsize.
- Check and reset population IDs if present
- Initialise population with the size from input parameter.
- Note that if the population is not allocated, all individuals that are not obtained from the data file are initialised random.
- Check and reset population IDs if present
- **MAIN_READ:** Cycle through the data lines using the non-advancing READLINE function and get the numerical genome data.
- All input file lines that contain data shorter than MIN_FILE_INP_LEN characters are ignored.
- Stripe away single and double quotes if any.
- An approximate upper bound for the number of data fields in the current record is len_trim(line_↩ data_buff)/MIN_FIELD), so we can now allocate the temporary array of substrings. This is also the total number of variables including the first two non-data records.
- Split the input string buffer line_data_buff into an array of individual strings representing each file data field. The number of such non-empty strings (fields) is given by line_data_nflds.

- **N_GCOLS:** Check if `line_data_nflds` obtained from CSV data coincides with the model (`this` data structure) as defined by the number of chromosomes, ploidy, chromosome length and N of additive components:
  - commondata::n_chromosomes,
  - commondata::chromosome_ploidy
  - commondata::len_chromosomes
  - commondata::additive_comps
- We have to allocate the array of row values and initialise it for correct processing by the `VALUE` subroutine.
- first value from the file row is `ID_NUM`
- second value from the file row is `AGENT_NAME`.
- now read a single row of the genome data for this (`icase`) individual agent, note that if any data in the file are abridged, random values are generated within the appropriate genome limits (commondata::allelerange_min, commondata::allelerange_max).
- Finally, split (parse) the single line array for the `icase`-s individual into the genome data structure.
- **FILL_EXTRA:** If the number of rows in the data file read is smaller than the population size commondata::popsize, the remaining data may need to be filled with some default values. The flag `missing_random` determines to initialise all the remaining individuals in the population as random.

Initialise all individuals of the population

- first, call `init to object individual(i)

Definition at line 2040 of file m_popul.f90.
Here is the call graph for this function:



8.11.3.25 population_save_data_memory()

```
subroutine the_population::population_save_data_memory (
             class(population), intent(in) this,
             character(len=*), intent(in) csv_file_name,
             logical, intent(out), optional is_success )
```

Save the perceptual and emotional memory stack data of all agents in this population to a CSV file.

Parameters

in	*csv_file_name*	csv_file_name is the name of the CSV output file.
out	*is_success*	is_success Flag showing that data save was successful (if TRUE).

8.11.3.25.1 Implementation details

8.11.3.25.1.1 Notable variables

- **COLUMNS_PERC** defines the column names for all components of the perceptual memory stack. They must agree with the components of perceptual memory: the_neurobio::memory_perceptual

COLUMNS_EMOT defines the column name for all components of the emotional memory stack. They must agree with the components of emotional memory: the_neurobio::memory_emotional.

8.11.3.25.1.2 Preliminary First, the file `csv_file_name` is opened for writing.

8.11.3.25.1.3 Build column names The maximum length of the data record is calculated from three components

- individual IDs: numeric ID and label;
- perceptual memory components from `COLUMNS_PERC` array for commondata::history_size_perception steps;
- emotional memory components from `COLUMNS_EMOT` array for commondata::history_size_motivation steps.

Note that, because the record also adds separators, such as comma and possibly double quotes, the number of columns multiplied by three is added to this value.
The first record of the data that contains the column names is now being "appended" into the complete record and written to the file. The maximum length of this record is calculated above.
The next portion is composed of the perceptual memory columns from `COLUMNS_PERC` array for each of the commondata::history_size_perception steps in the memory.

Note

Note that the order of data is: (1) perception component, (2) history steps: `PERC_LIGHT01, PERC_LIGHT02, PERC_LIGHT03, ... PERC_DEPTH01, PERC_DEPTH02, PERC_DEPTH03, ...`

The third portion is composed of the emotional memory columns from `COLUMNS_EMOT` array for each of the commondata::history_size_motivation steps in the memory.

Note

Note that the order of data is: (1) motivation component, (2) history steps: `MOTIV_HUNGER01, MOTIV_HUNGER02, MOTIV_HUNGER03, ... MOTIV_AVOIDPAS01, MOTIV_AVOIDPAS02, MOTIV_AVOIDPAS03, ...`.

After this step, the first record of column names is ready to be written to the file.

8.11.3.25.1.4 Write the numerical data The actual data are written in the same order as above, looping over all individual agents in this population.
The maximum record length `record_csv_max_length` is here the same as for writing the column headers, it is assumed that any numeric value in the data matrix occupies less than commondata::label_length characters.
So, for each agent, the following data are written with full history:

- ID data: numeric ID and the string label;
- Perceptual memory components;
- Emotional memory components;
- GOS memory components, note here that `gos_main` is a text value that is undefined (empty string) at the initialisation stage;

Each complete record is written to the file as it is built.
Once all individuals are saved, the file is closed.
The CSV output data file can be optionally compressed with the commondata::cmd_zip_output command if commondata::is_zip_outputs is set to TRUE.
Definition at line 2420 of file m_popul.f90.

8.11.3.26 population_save_data_movements()

```
subroutine the_population::population_save_data_movements (
            class(population), intent(in) this,
            character(len=*), intent(in) csv_file_name,
            logical, intent(out), optional is_success )
```

Save the latest movement history of all agents. This method makes use of the the_environment::spatial_moving::history structure that saves latest movements of each agent.

Parameters

in	*csv_file_name*	csv_file_name is the name of the CSV output file.
out	*is_success*	is_success Flag showing that data save was successful (if TRUE).

8.11.3.26.1 Implementation notes The maximum length of the data record is calculated from three components: (1) commondata::history_size_spatial $*$ 3 columns of X, Y, and depth coordinates plus (2) the same number of separators for these data columns (assuming 3 characters) plus (3) two additional columns that contain the numeric ID of the agent and its string label.
First, the file `csv_file_name` is opened for writing.

8.11.3.26.1.1 Build column names The first record of the data that contains the column names is now being built. The maximum length of this record is calculated above. First thing to do is to cleanup the record string.
After this, the first record is built by appending components.
The first two columns contain the identifiers for each agent:

- numeric ID of the agent
- test string label ("name") of the agent

All remaining columns are built in a loop, by triplets: "X", "Y", "Depth" for each step of the history, up to commondata::history_size_spatial triplets.
Now the first record of column names is ready to be written to the file.

8.11.3.26.1.2 Write the numerical data The actual data are written in the same order as above, looping over all individual agents in this population.

- The record string is cleaned;
- ID data appended: numeric ID and the string label;
- Movement history triplets (x, y, depth) for all commondata::history_size_spatial are appended to the record string in a loop.

Each complete record is written to the file as it is built.
Once all individuals are saved, the file is closed.
The CSV output data file can be optionally compressed with the commondata::cmd_zip_output command if commondata::is_zip_outputs is set to TRUE.
Definition at line 2634 of file m_popul.f90.

8.11.3.27 population_save_data_behaviours()

```
subroutine the_population::population_save_data_behaviours (
            class(population), intent(in) this,
            character(len=*), intent(in) csv_file_name,
            logical, intent(out), optional is_success )
```

Save the behaviours history stack the_neurobio::behaviour::history_behave for all agents.

Parameters

in	*csv_file_name*	csv_file_name is the name of the CSV output file.
out	*is_success*	is_success Flag showing that data save was successful (if TRUE).

8.11.3.27.1 Implementation notes The maximum length of the data record is calculated from three components: (1) commondata::history_size_behaviours labels of behaviours plus (2) the same number of separators for these data columns (assuming 3 characters) plus (3) two additional columns that contain the numeric ID of the agent and its string label.
First, the file `csv_file_name` is opened for writing.

8.11.3.27.1.1 Build column names The first record of the data that contains the column names is now being built. The maximum length of this record is calculated above. First thing to do is to cleanup the record string.
After this, the first record is built by appending components.
The first two columns contain the identifiers for each agent:

- numeric ID of the agent
- test string label ("name") of the agent

All remaining columns are built in a loop for each step of the history, up to commondata::history_size_behaviours steps back in history.
Now the first record of column names is ready to be written to the file.

8.11.3.27.1.2 Write the numerical data The actual data are written in the same order as above, looping over all individual agents in this population.

- The record string is cleaned;
- ID data appended: numeric ID and the string label;
- Behaviour history for all commondata::history_size_behaviours steps is appended to the record string in a loop.

Each complete record is written to the file as it is built.
Once all individuals are saved, the file is closed.
The CSV output data file can be optionally compressed with the commondata::cmd_zip_output command if commondata::is_zip_outputs is set to TRUE.
Definition at line 2761 of file m_popul.f90.

8.11.3.28 population_preevol_fitness_calc()

```
pure subroutine the_population::population_preevol_fitness_calc (
            class(population), intent(inout) this )
```

Calculate fitness for the pre-evolution phase of the genetic algorithm. **Pre-evolution** is based on selection for a simple criterion without explicit reproduction etc. The criterion for selection at this phase is set by the integer the_individual::individual_agent::fitness component. This procedure provides a whole-population wrapper for the the_individual::individual_agent::fitness_calc() function.

Warning

Note that fitness here is actually an inverse of the fitness: the higher its value, the worse fitting is the agent.

Definition at line 2888 of file m_popul.f90.

8.11.3.29 population_ga_reproduce_max()

```
pure integer function the_population::population_ga_reproduce_max (
            class(population), intent(in) this )
```

Determine the number of parents that have fitness higher than the minimum acceptable value. Also, only alive agents are included into the reproducing number.

Note

This procedure is used in the fixed explicit fitness genetic algorithm, see the_evolution::generations_loop_ga().

- MIN_FITNESS is the normal limit of the fitness value for inclusion into the reproducing elite group. See also the_individual::individual_agent::individual_preevol_fitness_calc().
- MIN_GA_REPRODUCE is the minimum GA_REPRODUCE, the final value cannot be smaller than. It is set as the minimum proportion commondata::ga_reproduce_min_prop of the commondata::popsize. However, it cannot be smaller than the absolute minimum commondata::ga_reproduce_n_min.

Definition at line 2901 of file m_popul.f90.

8.11.3.30 population_ga_mutation_rate_adaptive()

```
real(srp) function the_population::population_ga_mutation_rate_adaptive (
            class(population), intent(in) this,
            real(srp), intent(in) baseline,
            real(srp), intent(in), optional maxvalue )
```

This function implements adaptive mutation rate that *increases* as the population size *reduces*.

Parameters

in	*baseline*	baseline baseline mutation rate
in	*maxvalue*	maxvalue maximum mutation rate

Returns

The adjusted adaptive mutation rate.

8.11.3.30.1 Implementation notes

8.11.3.30.1.1 Notable variables and parameters

- mutationrate_max – the maximum limit to the mutation rate. Can be obtained from the optional parameter maxvalue. The function returns its value at the lowest population size.

  Remarks

  It is actually local copy of optional maxvalue parameter.

MUTATIONRATE_MAX_DEF – the default maximum limit to the mutation rate mutationrate_max if maxvalue is not provided.

- n_base_point – this is the base value for calculation of the adaptive mutation rate. It can be the number of agents alive or the number of agents that have grown.
- mutation_grid_abscissa and mutation_grid_ordinate are the arrays that define the interpolation grid for the adaptive mutation rate.
- MIN_GROWING a parameter setting the minimum number of growing agents in the population. If their actual number is smaller, the mutation rate further incremented by a factor set by the parameter
- NON_GROW_INCREMENT is an increment factor for the mutation rate in case the number of growing agents is below the lower limit MIN_GROWING.

8.11.3.30.1.2 Procedure First, calculate the base point `n_base_point`. This value is the base for the calculation of the adaptive mutation rate.

- If this number reduces, mutation rate increases up to `mutationrate_max`

The `n_base_point` can be:

- Number of agents that are **alive**: n_base_point = count(thisindividualis_alive())
- Number of agents that have **grown**: n_base_point = count(thisindividualget_mass() > & thisindividualget↩ _mass_birth())
- If the mutation rate is based on the number of alive agents, the grid abscissa `mutation_grid_abscissa` is an array with three elements:
  - minimum full population size
  - 1/2 of the full population size commondata::popsize
  - full commondata::popsize
- If, on the other hand, adaptive mutation rate is based on the number of agents that have grown, abscissa is set to specific arbitrary numbers that seem more or less optimal for the model performance.

The grid ordinate is defined as

- the maximum mutation rate limit `mutationrate_max`
- a small middle value that is calculated as 1/4 of the range between the maximum (`MUTATIOlNRATE_MAX`) and the minimum (`baseline`) mutation values; the latter value increments the minimum `baseline`.
- the lowest baseline value that is set by the `baseline` dummy parameter.

Adaptive mutation rate is calculated based on the DDPINTERPOL() procedure with the grid array set by `mutation_grid_abscissa` and `mutation_grid_ordinate` and the interpolation value set by the number of agents that are the_genome::individual_genome::is_alive() in the population. An example pattern of the adaptive mutation rate function is plotted below. Here P_{max} is the maximum mutation rate defined by `mutationrate_max` and P_b is the baseline (normal, low) mutation rate defined by the `baseline` parameter.

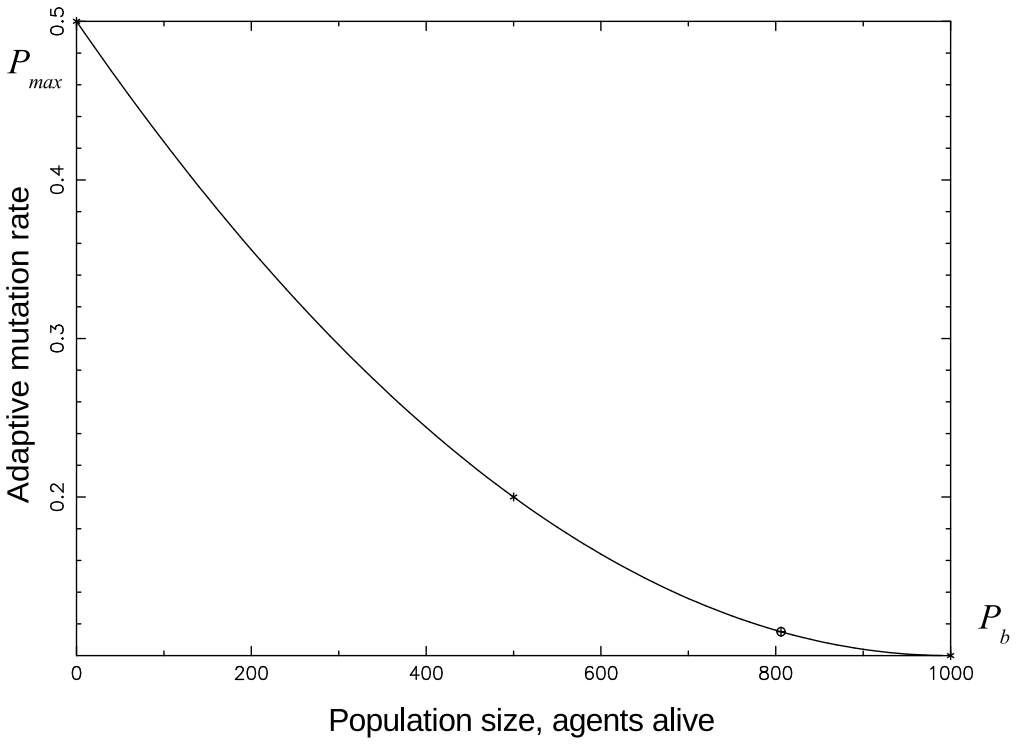


Figure 8.34 Adaptive mutation rate

If the number of agents that had grown from their birth mass is less than the minimum number (MIN_GROWING), mutation rate is incremented by a fixed factor NON_GROW_INCREMENT. However, it is still forced to be within the range [baseline, mutationrate_max].
Interpolation plots can be saved in the debug mode using this plotting command: commondata::debug_interpolate_plot_save().

Warning

Involves **huge** number of plots, should normally be disabled.

Definition at line 2936 of file m_popul.f90.

8.11.4 Variable Documentation

8.11.4.1 modname

character (len=*), parameter, private the_population::modname = "(THE_POPULATION)" [private]
Definition at line 25 of file m_popul.f90.

8.11.4.2 global_ind_n_eaten_by_predators

integer, public the_population::global_ind_n_eaten_by_predators
Global indicator variable that keeps the number of agents that have died as a consequence of predatory attacks. All other dies are therefore caused by starvation.

Note

Note that this variable is initialised to zero at the start of each generation in GENERATIONS_PREEVOL named loop in the_evolution::generations_loop_ga().

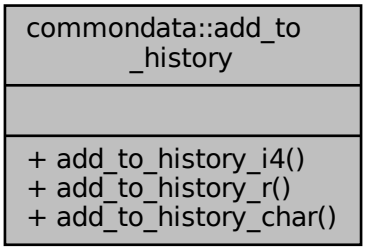
Definition at line 183 of file m_popul.f90.

Chapter 9

Data Type Documentation

9.1 commondata::add_to_history Interface Reference

Simple history stack function, add to the end of the stack. We need only to add components on top (end) of the stack and retain `HISTORY_SIZE_SPATIAL` elements of the prior history (for a spatial moving object). The stack works as follows, assuming 100 and 200 are added:
[1 2 3 4 5 6 7 8 9 10]
[2 3 4 5 6 7 8 9 10 **100**]
[3 4 5 6 7 8 9 10 100 **200**].
Collaboration diagram for commondata::add_to_history:



Public Member Functions

- pure subroutine add_to_history_i4 (history_array, add_this)

  Simple history stack function, add to the end of the stack. We need only to add components on top of the stack and retain `commondata::history_size_spatial` elements of the prior history (for a spatial moving object). The stack works as follows, assuming 100 and 200 are added:
  [1 2 3 4 5 6 7 8 9 10];
  [2 3 4 5 6 7 8 9 10 100];
  [3 4 5 6 7 8 9 10 100 200].
- pure subroutine add_to_history_r (history_array, add_this)

  Simple history stack function, add to the end of the stack. We need only to add components on top of the stack and retain `commondata::history_size_spatial` elements of the prior history (for a spatial moving object).
- pure subroutine add_to_history_char (history_array, add_this)

  Simple history stack function, add to the end of the stack. We need only to add components on top of the stack and retain `commondata::history_size_spatial` elements of the prior history.

9.1.1 Detailed Description

Simple history stack function, add to the end of the stack. We need only to add components on top (end) of the stack and retain HISTORY_SIZE_SPATIAL elements of the prior history (for a spatial moving object). The stack works as follows, assuming 100 and 200 are added:
[1 2 3 4 5 6 7 8 9 10]
[2 3 4 5 6 7 8 9 10 **100**]
[3 4 5 6 7 8 9 10 100 **200**].
Definition at line 5292 of file m_common.f90.

9.1.2 Member Function/Subroutine Documentation

9.1.2.1 add_to_history_i4()

```
pure subroutine commondata::add_to_history::add_to_history_i4 (
             integer, dimension(:), intent(inout) history_array,
             integer, intent(in) add_this )
```

Simple history stack function, add to the end of the stack. We need only to add components on top of the stack and retain commondata::history_size_spatial elements of the prior history (for a spatial moving object). The stack works as follows, assuming 100 and 200 are added:
[1 2 3 4 5 6 7 8 9 10];
[2 3 4 5 6 7 8 9 10 100];
[3 4 5 6 7 8 9 10 100 200].

Parameters

history_array	Integer array that keeps the history.
add_this	we add this element to the end of the history array.

Note

This is the integer type version.

Definition at line 7299 of file m_common.f90.

9.1.2.2 add_to_history_r()

```
pure subroutine commondata::add_to_history::add_to_history_r (
             real(srp), dimension(:), intent(inout) history_array,
             real(srp), intent(in) add_this )
```

Simple history stack function, add to the end of the stack. We need only to add components on top of the stack and retain commondata::history_size_spatial elements of the prior history (for a spatial moving object).

Parameters

history_array	Integer array that keeps the history.
add_this	we add this element to the end of the history array.

Note

This is the real type version.

Definition at line 7324 of file m_common.f90.

9.1.2.3 add_to_history_char()

```
pure subroutine commondata::add_to_history::add_to_history_char (
            character(*), dimension(:), intent(inout) history_array,
            character(*), intent(in) add_this )
```

Simple history stack function, add to the end of the stack. We need only to add components on top of the stack and retain commondata::history_size_spatial elements of the prior history.

Parameters

history_array	Integer array that keeps the history.
add_this	we add this element to the end of the history array.

Note

This is the character string type version

Definition at line 7348 of file m_common.f90.
The documentation for this interface was generated from the following file:

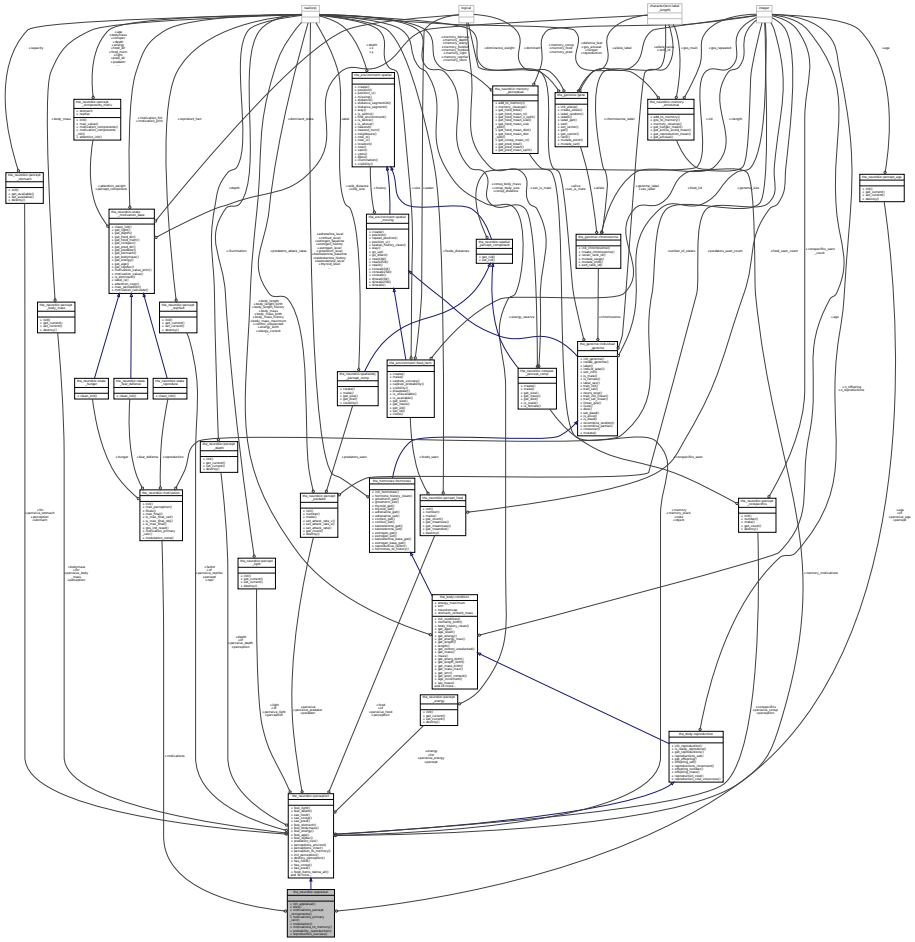
- m_common.f90

9.2 the_neurobio::appraisal Type Reference

The **appraisal** level. At this level, perception objects are feed into the commondata::gamma2gene() sigmoid function and the neuronal responses are obtained at the output. Neuronal responses for different perception objects are then summed up and the promary motivation values are obtained. Following this, modulation alters some of the primary motivation values resulting in the final motivation values. See "From perception to GOS" for an overview.

Inheritance diagram for the_neurobio::appraisal:

Collaboration diagram for the_neurobio::appraisal:



Public Member Functions

- procedure, public init_appraisal => appraisal_init_zero_cleanup_all

  Initialise and cleanup all appraisal object components and sub-objects. See `the_neurobio::appraisal_init_zero_cleanup`

- procedure, public dies => appraisal_agent_set_dead

  *Set the individual to be **dead**. This method overrides the the_genome::individual_genome::dies() method, nullifying all reproductive and neurobiological and behavioural objects. However, this function does not deallocate the individual agent object, this may be a separate destructor function. The* `dies` *method is implemented at the following levels of the agent object hierarchy (upper overrides the lower level):*

- procedure, public motivations_percept_components => appraisal_perceptual_comps_motiv_neur_response_calculate

  *Calculate **perception components** for each of the motivational state component.*

- procedure, public motivations_primary_calc => appraisal_primary_motivations_calculate

  *Calculate **primary motivation values** of the agent by summing up the perception components of each motivation state.*

- procedure, public modulation => appraisal_motivation_modulation_non_genetic

  *Calculate the **final motivation values** after **modulation**.*

- procedure, public motivations_to_memory => appraisal_add_final_motivations_memory

  Add individual final emotional state components into the emotional memory stack. See `the_neurobio::appraisal_add_final_`

- procedure, public probability_reproduction => reproduce_do_probability_reproduction_calc

  Calculate the probability of successful reproduction for `this` *agent in its current state.*

- procedure, public reproduction_success => reproduction_success_stochast

  *Determine a stochastic outcome of **this** agent reproduction. Returns TRUE if the agent has reproduced successfully. See* `the_neurobio::reproduction_success_stochast()`.

Public Attributes

- type(motivation) motivations

  The appraisal component plugs-in the different motivational/emotional objects.
- type(memory_emotional) memory_motivations

  The emotional state memory stack object.

9.2.1 Detailed Description

The **appraisal** level. At this level, perception objects are feed into the commondata::gamma2gene() sigmoid function and the neuronal responses are obtained at the output. Neuronal responses for different perception objects are then summed up and the promary motivation values are obtained. Following this, modulation alters some of the primary motivation values resulting in the final motivation values. See "From perception to GOS" for an overview.
Definition at line 1222 of file m_neuro.f90.

9.2.2 Member Function/Subroutine Documentation

9.2.2.1 init_appraisal()

`procedure, public the_neurobio::appraisal::init_appraisal`
Initialise and cleanup all appraisal object components and sub-objects. See `the_neurobio::appraisal_init_zero_clea`
Definition at line 1231 of file m_neuro.f90.

9.2.2.2 dies()

`procedure, public the_neurobio::appraisal::dies`
Set the individual to be **dead**. This method overrides the the_genome::individual_genome::dies() method, nullifying all reproductive and neurobiological and behavioural objects. However, this function does not deallocate the individual agent object, this may be a separate destructor function. The `dies` method is implemented at the following levels of the agent object hierarchy (upper overrides the lower level):

- the_genome::individual_genome::dies();
- the_neurobio::appraisal::dies();
- the_neurobio::gos_global::dies();
- the_individual::individual_agent::dies().

See `the_individual::appraisal_agent_set_dead()`.
Definition at line 1245 of file m_neuro.f90.

9.2.2.3 motivations_percept_components()

`procedure, public the_neurobio::appraisal::motivations_percept_components`
Calculate **perception components** for each of the motivational state component.
Initialise motivational states from perception objects through the neuronal response function. @important We initialise here all the **perception components** (`the_neurobio::percept_components_motiv`) for every **motivational state** component. See `the_neurobio::appraisal_perceptual_comps_motiv_neur_response_ca`
Definition at line 1254 of file m_neuro.f90.

9.2.2.4 motivations_primary_calc()

procedure, public the_neurobio::appraisal::motivations_primary_calc
Calculate **primary motivation values** of the agent by summing up the perception components of each motivation state.
Here it is just wrapper to the the_neurobio::motivation -bound procedure motivation_primary_calc. See the_neurobio::appraisal_primary_motivations_calculate().
Definition at line 1261 of file m_neuro.f90.

9.2.2.5 modulation()

procedure, public the_neurobio::appraisal::modulation
Calculate the **final motivation values** after **modulation**.
Perform developmental and/or genetic modulation of primary motivations that result in the final motivation values.

Note

Genetic modulation backend the_neurobio::appraisal_motivation_modulation_genetic() is bound to the agent rather than the_neurobio::motivation. See the_neurobio::appraisal_motivation_modulation_non_genetic

Definition at line 1270 of file m_neuro.f90.

9.2.2.6 motivations_to_memory()

procedure, public the_neurobio::appraisal::motivations_to_memory
Add individual final emotional state components into the emotional memory stack. See the_neurobio::appraisal_add_fin
Definition at line 1275 of file m_neuro.f90.

9.2.2.7 probability_reproduction()

procedure, public the_neurobio::appraisal::probability_reproduction
Calculate the probability of successful reproduction for this agent in its current state.

Note

Note that this function is defined and bound to the_neurobio::appraisal but used in the_neurobio::reproduce behavioural component class. See the_neurobio::reproduce_do_probability_rep

Definition at line 1283 of file m_neuro.f90.

9.2.2.8 reproduction_success()

procedure, public the_neurobio::appraisal::reproduction_success
Determine a stochastic outcome of **this** agent reproduction. Returns TRUE if the agent has reproduced successfully.
See the_neurobio::reproduction_success_stochast().
Definition at line 1288 of file m_neuro.f90.

9.2.3 Member Data Documentation

9.2.3.1 motivations

type(motivation) the_neurobio::appraisal::motivations
The appraisal component plugs-in the different motivational/emotional objects.
Definition at line 1225 of file m_neuro.f90.

9.2.3.2 memory_motivations

```
type(memory_emotional) the_neurobio::appraisal::memory_motivations
```

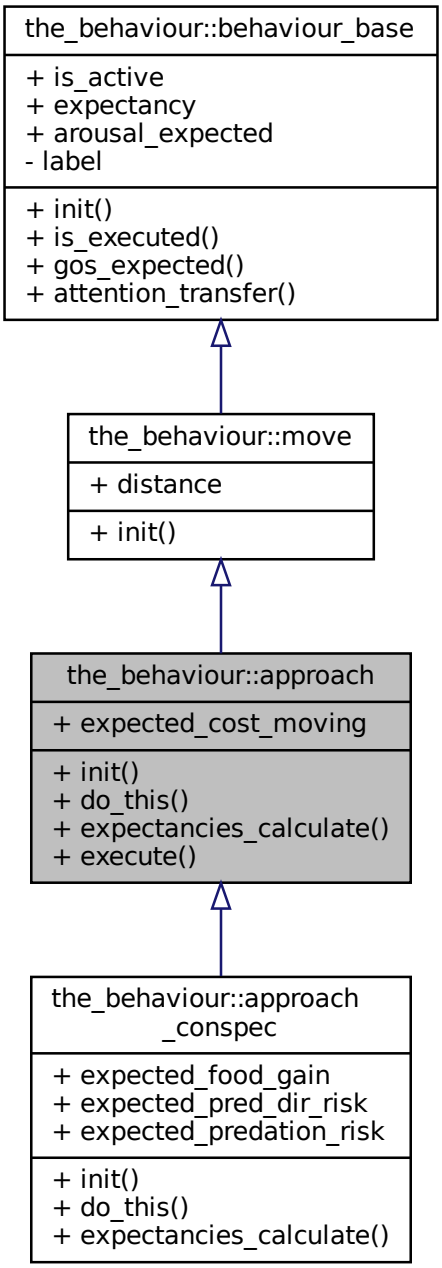
The emotional state memory stack object.
Definition at line 1227 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

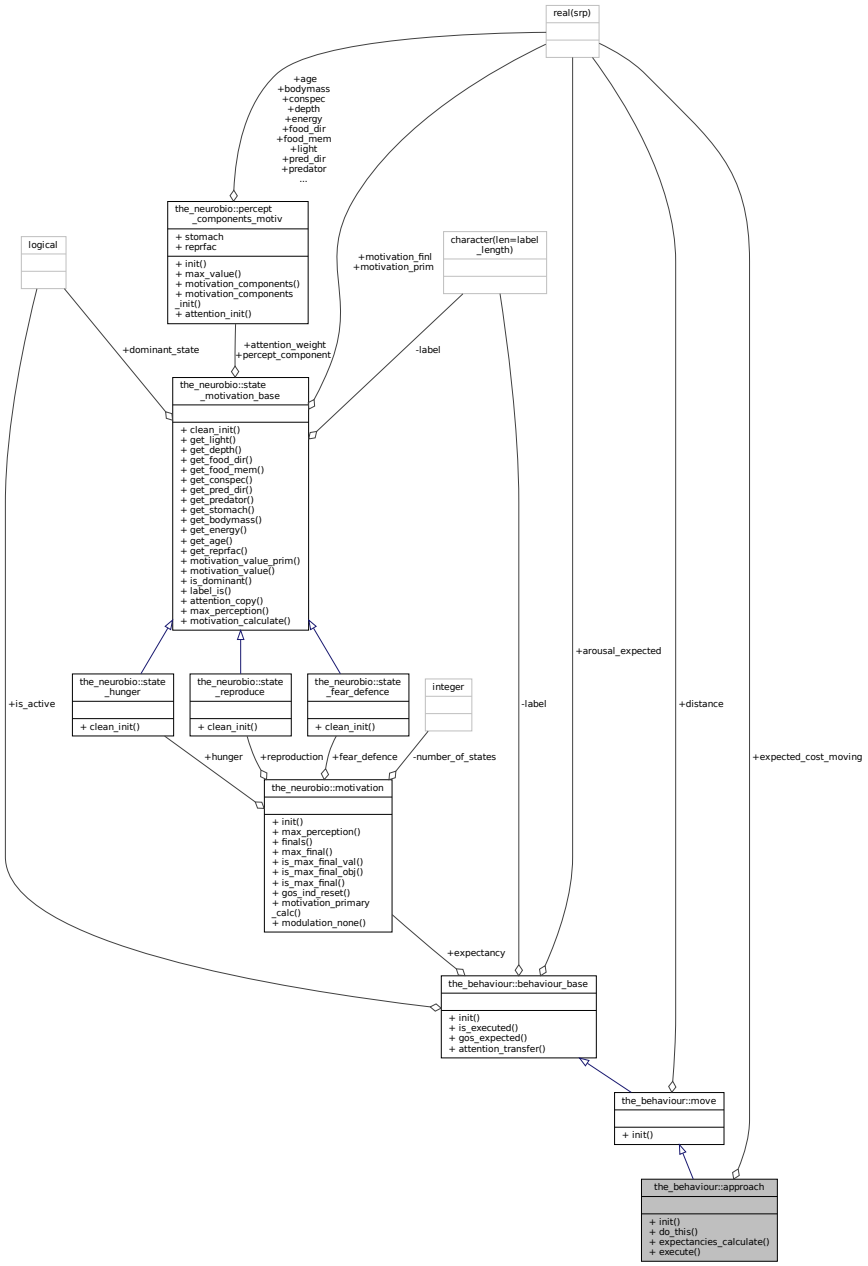
9.3 the_behaviour::approach Type Reference

Approach an arbitrary spatial object is a directed movement to an arbitrary the_environment::spatial class target object.

Inheritance diagram for the_behaviour::approach:



Collaboration diagram for the_behaviour::approach:



Public Member Functions

- procedure, public init => approach_spatial_object_init_zero

  *Initialise the **approach** behaviour component to a zero state. Approach is a generic type but not abstract. See* `the_behaviour::approach_spatial_object_init_zero()`.

- procedure, public do_this => approach_do_this

  The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here the_environment::approach). See `the_behaviour::approach_do_this()`.

- procedure, public expectancies_calculate => approach_motivations_expect

the_behaviour::approach::expectancies_calculate() (re)calculates motivations from fake expected perceptions following from the procedure approach::do_this() => the_behaviour::approach_do_this(). See the_behaviour::approach_motivations_expect().

- procedure, public execute => approach_do_execute

  Execute this behaviour component "approach" by this_agent agent. See the_behaviour::approach_do_execute().

Public Attributes

- real(srp) expected_cost_moving

  The body mass cost of movement; depends on the distance.

9.3.1 Detailed Description

Approach an arbitrary spatial object is a directed movement to an arbitrary the_environment::spatial class target object.
Definition at line 290 of file m_behav.f90.

9.3.2 Member Function/Subroutine Documentation

9.3.2.1 init()

procedure, public the_behaviour::approach::init
Initialise the **approach** behaviour component to a zero state. Approach is a generic type but not abstract. See the_behaviour::approach_spatial_object_init_zero().
Definition at line 302 of file m_behav.f90.

9.3.2.2 do_this()

procedure, public the_behaviour::approach::do_this
The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here the_environment::approach). See the_behaviour::approach_do_this().
Definition at line 309 of file m_behav.f90.

9.3.2.3 expectancies_calculate()

procedure, public the_behaviour::approach::expectancies_calculate
the_behaviour::approach::expectancies_calculate() (re)calculates motivations from fake expected perceptions following from the procedure approach::do_this() => the_behaviour::approach_do_this().
See the_behaviour::approach_motivations_expect().
Definition at line 314 of file m_behav.f90.

9.3.2.4 execute()

procedure, public the_behaviour::approach::execute
Execute this behaviour component "approach" by this_agent agent. See the_behaviour::approach_do_execute().
Definition at line 317 of file m_behav.f90.

9.3.3 Member Data Documentation

9.3.3.1 expected_cost_moving

`real(srp) the_behaviour::approach::expected_cost_moving`
The body mass cost of movement; depends on the distance.

Note

Note that such class attributes as `expected_food_gain`, `expected_food_gain expected_pred_dir_risk` `expected_predation_risk` should be implemented in specific derived subclasses of the_behaviour::approach, e.g. the_behaviour::approach_conspec.

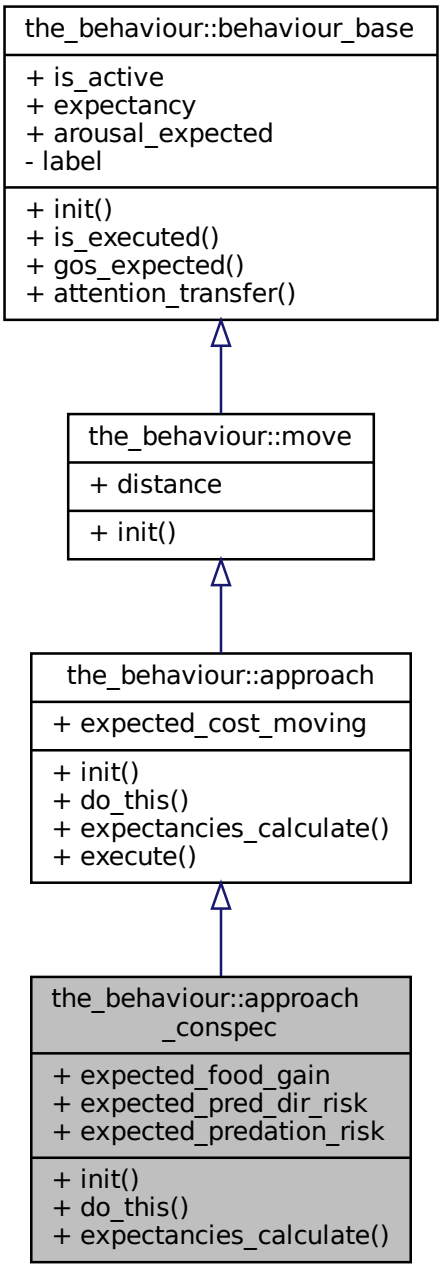
Definition at line 297 of file m_behav.f90.
The documentation for this type was generated from the following file:

- m_behav.f90

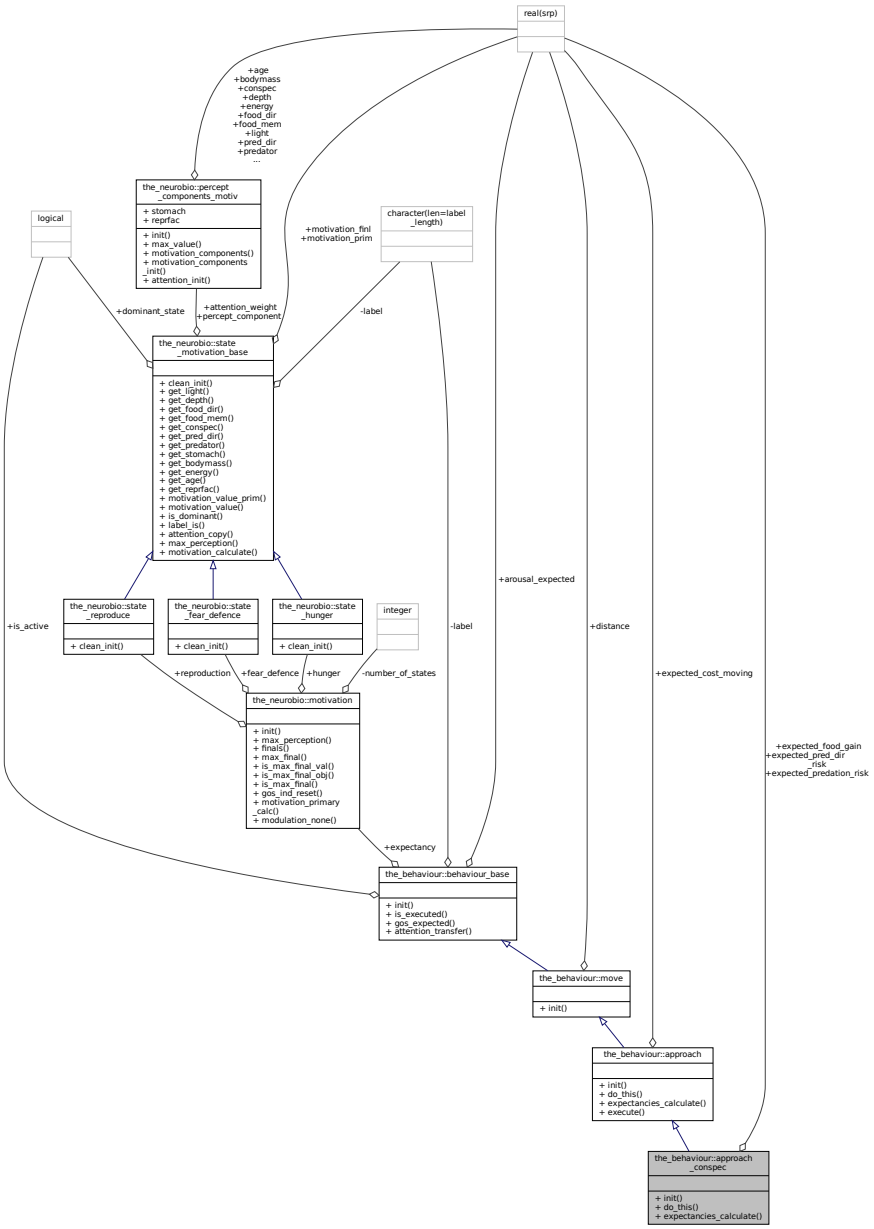
9.4 the_behaviour::approach_conspec Type Reference

Approach conspecifics is directed movement towards a conspecific.

Inheritance diagram for the_behaviour::approach_conspec:



Collaboration diagram for the_behaviour::approach_conspec:



Public Member Functions

- procedure, public init => approach_conspecifics_init_zero

  *Initialise the **approach conspecific** behaviour to a zero state. Approach conspecific is a special extension of the generic `the_behaviour::approach` behaviour. See `the_behaviour::approach_conspecifics_init_zero()`.*

- procedure, public do_this => approach_conspecifics_do_this

  The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here `APPROACH_CONSPES`). See `the_behaviour::approach_conspecifics_do_this()`.

- procedure, public expectancies_calculate => approach_conspecifics_motivations_expect

  `the_behaviour::approach_conspec::expectancies_calculate()` (re)calculates motivations from fake expected perceptions following from the procedure approach_conspec::do_this(). See `the_behaviour::approach_conspecifics_motivations_expect()`.

Public Attributes

- real(srp) expected_food_gain

  *The expected food gain (body mass increment) is always **null** for active escape.*
- real(srp) expected_pred_dir_risk

  The expected direct predation risk, from the nearest predator.
- real(srp) expected_predation_risk

  The expected general predation risk, i.e. the risk depending on the current number of predators in both the perception and memory stack.

9.4.1 Detailed Description

Approach conspecifics is directed movement towards a conspecific.

Note

The `execute` method for the_behaviour::approach_conspec uses the base class the_behaviour::approach::execute() method.

Definition at line 323 of file m_behav.f90.

9.4.2 Member Function/Subroutine Documentation

9.4.2.1 init()

`procedure, public the_behaviour::approach_conspec::init`
Initialise the **approach conspecific** behaviour to a zero state. Approach conspecific is a special extension of the generic `the_behaviour::approach` behaviour. See `the_behaviour::approach_conspecifics_init_zero()`.
Definition at line 337 of file m_behav.f90.

9.4.2.2 do_this()

`procedure, public the_behaviour::approach_conspec::do_this`
The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here `APPROACH_CONSPES`). See `the_behaviour::approach_conspecifics_do_this()`.
Definition at line 344 of file m_behav.f90.

9.4.2.3 expectancies_calculate()

`procedure, public the_behaviour::approach_conspec::expectancies_calculate`
`the_behaviour::approach_conspec::expectancies_calculate()` (re)calculates motivations from fake expected perceptions following from the procedure approach_conspec::do_this(). See `the_behaviour::approach_conspecifics_motivations_expect()`.
Definition at line 349 of file m_behav.f90.

9.4.3 Member Data Documentation

9.4.3.1 expected_food_gain

`real(srp) the_behaviour::approach_conspec::expected_food_gain`
The expected food gain (body mass increment) is always **null** for active escape.
Definition at line 326 of file m_behav.f90.

9.4.3.2 expected_pred_dir_risk

`real(srp) the_behaviour::approach_conspec::expected_pred_dir_risk`
The expected direct predation risk, from the nearest predator.
Definition at line 328 of file m_behav.f90.

9.4.3.3 expected_predation_risk

`real(srp) the_behaviour::approach_conspec::expected_predation_risk`
The expected general predation risk, i.e. the risk depending on the current number of predators in both the perception and memory stack.
Definition at line 331 of file m_behav.f90.
The documentation for this type was generated from the following file:

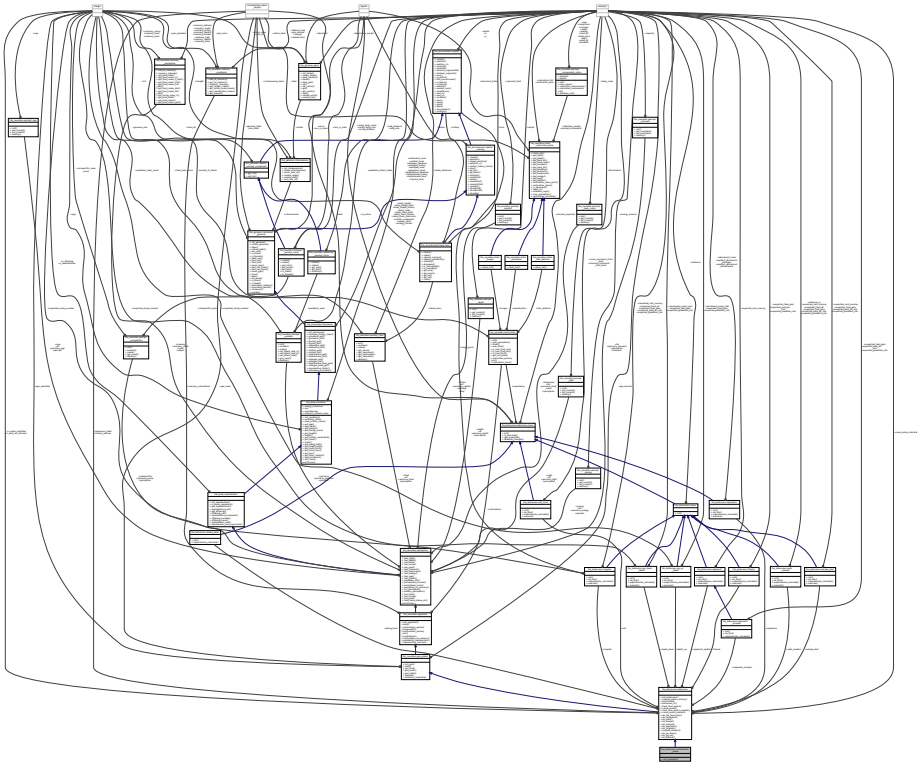
- m_behav.f90

9.5 the_behaviour::architecture_neuro Type Reference

This type is an "umbrella" for all the lower-level classes.

Inheritance diagram for the_behaviour::architecture_neuro:

Collaboration diagram for the_behaviour::architecture_neuro:



Public Member Functions

- procedure, public init_neurobio => neurobio_init_components

  Initialise neuro-biological architecture. See `the_behaviour::neurobio_init_components()` for implementation.

Additional Inherited Members

9.5.1 Detailed Description

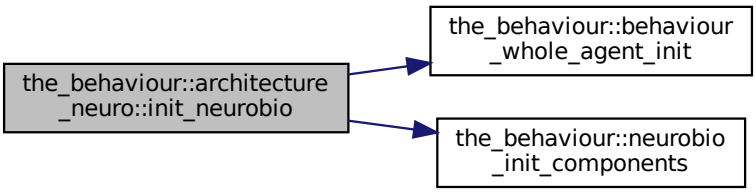
This type is an "umbrella" for all the lower-level classes.
Definition at line 627 of file m_behav.f90.

9.5.2 Member Function/Subroutine Documentation

9.5.2.1 init_neurobio()

`procedure, public the_behaviour::architecture_neuro::init_neurobio`
Initialise neuro-biological architecture. See `the_behaviour::neurobio_init_components()` for implementation.
Definition at line 632 of file m_behav.f90.

Here is the call graph for this function:

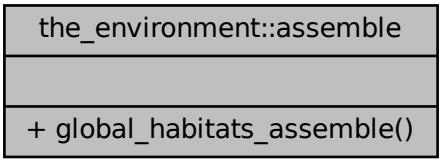


The documentation for this type was generated from the following file:

- m_behav.f90

9.6 the_environment::assemble Interface Reference

Interface to the procedure to **assemble** the global array of habitat objects the_environment::global_habitats_available from a list of separate habitat object components. This call.
Collaboration diagram for the_environment::assemble:



Public Member Functions

- subroutine global_habitats_assemble (habitat_1, habitat_2, habitat_3, habitat_4, habitat_5, habitat_↩ 6, habitat_7, habitat_8, habitat_9, habitat_10, habitat_11, habitat_12, habitat_13, habitat_14, habitat_15, habitat_16, habitat_17, habitat_18, habitat_19, habitat_20, reindex)

  Assemble the global habitats objects array the_environment::global_habitats_available from a list of separate habitat objects. This call.

9.6.1 Detailed Description

Interface to the procedure to **assemble** the global array of habitat objects the_environment::global_habitats_available from a list of separate habitat object components. This call.

```
assemble(hab_a, hab_b, hab_c)
```

is equivalent to

```
global_habitats_available = [ hab_a, hab_b, hab_c ]
```

See the_environment::global_habitats_assemble() for the backend implementation.
Definition at line 797 of file m_env.f90.

9.6.2 Member Function/Subroutine Documentation

9.6.2.1 global_habitats_assemble()

```
subroutine the_environment::assemble::global_habitats_assemble (
             type(habitat), intent(in), optional habitat_1,
             type(habitat), intent(in), optional habitat_2,
             type(habitat), intent(in), optional habitat_3,
             type(habitat), intent(in), optional habitat_4,
             type(habitat), intent(in), optional habitat_5,
             type(habitat), intent(in), optional habitat_6,
             type(habitat), intent(in), optional habitat_7,
             type(habitat), intent(in), optional habitat_8,
             type(habitat), intent(in), optional habitat_9,
             type(habitat), intent(in), optional habitat_10,
             type(habitat), intent(in), optional habitat_11,
             type(habitat), intent(in), optional habitat_12,
             type(habitat), intent(in), optional habitat_13,
             type(habitat), intent(in), optional habitat_14,
             type(habitat), intent(in), optional habitat_15,
             type(habitat), intent(in), optional habitat_16,
             type(habitat), intent(in), optional habitat_17,
             type(habitat), intent(in), optional habitat_18,
             type(habitat), intent(in), optional habitat_19,
             type(habitat), intent(in), optional habitat_20,
             logical, intent(in), optional reindex )
```

Assemble the global habitats objects array the_environment::global_habitats_available from a list of separate habitat objects. This call.

```
assemble(hab_a, hab_b, hab_c)
```

is equivalent to

```
global_habitats_available = [ hab_a, hab_b, hab_c ]
```

Note

But note that the `reindex` parameter allows automatic reindexing of the global array the_environment::global_habitats_available

Parameters

in	*habitat_1*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_2*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_3*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_4*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_5*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_6*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_7*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_8*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_9*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_10*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_11*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_12*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_13*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_14*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_15*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_16*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_17*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_18*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_19*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*habitat_20*	[inout] habitat_1, ... a list (up to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

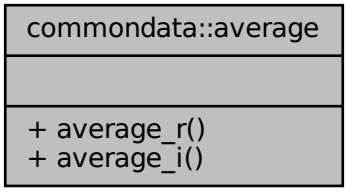
in	*reindex*	reindex logical flag to reindex the global joined food resource array (TRUE) linked to each of the habitats upon assemble. Default is **no** reindexing.

Definition at line 8701 of file m_env.f90.
The documentation for this interface was generated from the following file:

- m_env.f90

9.7 commondata::average Interface Reference

Calculate an average of an array excluding missing code values.
Collaboration diagram for commondata::average:



Public Member Functions

- pure real(srp) function average_r (array_in, missing_code, undef_ret_null)

  Calculate an average value of a real array, excluding MISSING values.
- pure real(srp) function average_i (array_in, missing_code, undef_ret_null)

  Calculate an average value of an integer array, excluding MISSING values.

9.7.1 Detailed Description

Calculate an average of an array excluding missing code values.
Definition at line 5491 of file m_common.f90.

9.7.2 Member Function/Subroutine Documentation

9.7.2.1 average_r()

```
pure real(srp) function commondata::average::average_r (
            real(srp), dimension(:), intent(in) array_in,
            real(srp), intent(in), optional missing_code,
            logical, intent(in), optional undef_ret_null )
```

Calculate an average value of a real array, excluding MISSING values.

Parameters

vector_in	The input data vector
missing_code	Optional parameter setting the missing data code, to be excluded from the calculation of the mean.
undef_ret_null	Optional parameter, if TRUE, the function returns zero rather than undefined if the sample size is zero.

Returns

The mean value of the vector.

Note

This is a real array version.

Definition at line 5952 of file m_common.f90.

9.7.2.2 average_i()

```
pure real(srp) function commondata::average::average_i (
             integer, dimension(:), intent(in) array_in,
             integer, intent(in), optional missing_code,
             logical, intent(in), optional undef_ret_null )
```

Calculate an average value of an integer array, excluding MISSING values.

Returns

The mean value of the vector

Parameters

vector_in	The input data vector
missing_code	Optional parameter setting the missing data code, to be excluded from the calculation of the mean.
undef_ret_null	Optional parameter, if TRUE, the function returns zero rather than undefined if the sample size is zero.

Note

This is an integer array version.

Definition at line 6022 of file m_common.f90.
The documentation for this interface was generated from the following file:

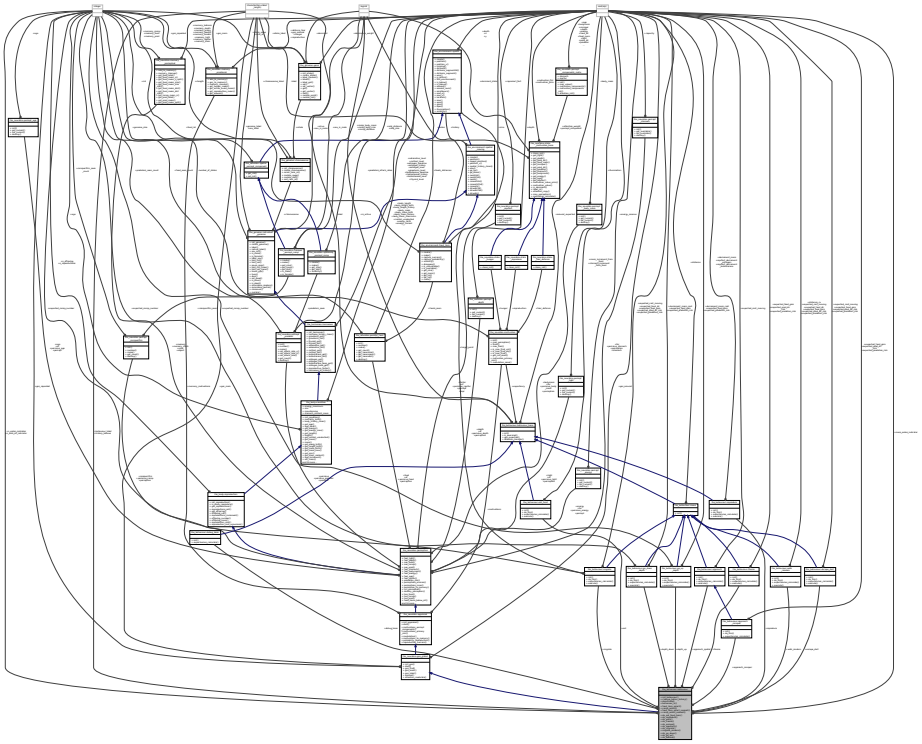
- m_common.f90

9.8 the_behaviour::behaviour Type Reference

The behaviour of the agent is defined by the the_behaviour::behaviour class. This class defines the *behavioural repertoire* of the agent. Each of the components of the behavioural repertoire (behaviour object) is defined as a separate independent class with its own *self* parameter. However, the agent which performs the behaviour (the *actor agent*) is included as the first non-self parameter into the behaviour component methods.

Inheritance diagram for the_behaviour::behaviour:

Collaboration diagram for the_behaviour::behaviour:



Public Member Functions

- procedure, public init_behaviour => behaviour_whole_agent_init

  The subroutines contained define what the agent really does, i.e. implements the actual behavioural repertoire components.
- procedure, public cleanup_behav_history => behaviour_cleanup_history

  Cleanup the behaviour history stack for the agent. See `the_behaviour::behaviour_cleanup_history()`.
- procedure, public deactivate => behaviour_whole_agent_deactivate

  Deactivate all behaviour units that compose the behaviour repertoire of the agent. See `the_behaviour::behaviour_whole_age`
- procedure, public behaviour_is => behaviour_get_behaviour_label_executing

  Obtain the label of the currently executing behaviour for the `this` *agent. See* `the_behaviour::behaviour_get_behaviour_l`
- procedure, public food_item_select => behaviour_select_food_item

  Select the optimal food item among (possibly) several ones that are available in the ***perception object*** *of the agent. See* `the_behaviour::behaviour_select_food_item()`.
- procedure, public consp_select => behaviour_select_conspecific

  Select the optimal conspecific among (possibly) several ones that are available in the ***perception object*** *of the agent. See* `the_behaviour::behaviour_select_conspecific()`.
- procedure, public food_item_select_nearest => behaviour_select_food_item_nearest

  Select the nearest food item among (possibly) several ones that are available in the perception object. See `the_behaviour::behaviour_select_food_item_nearest()`.
- procedure, public consp_select_nearest => behaviour_select_conspecific_nearest

  Select the nearest conspecific among (possibly) several ones that are available in the perception object. See `the_behaviour::behaviour_select_conspecific_nearest()`.
- procedure, public do_eat_food_item => behaviour_do_eat_food_item

  Eat a food item(s) that are found in the perception object. See `the_behaviour::behaviour_do_eat_food_item()`.
- procedure, public do_reproduce => behaviour_do_reproduce

  Reproduce based on the `this` *agent's current state. See* `the_behaviour::behaviour_do_reproduce()`.
- procedure, public do_walk => behaviour_do_walk

Perform a random Gaussian walk to a specific average distance with certain variance (set by the CV). See `the_behaviour::behaviour_do_walk()`.

- procedure, public do_freeze => behaviour_do_freeze

  Perform (execute) the the_behaviour::freeze behaviour. See `the_behaviour::behaviour_do_freeze()`.
- procedure, public do_escape => behaviour_do_escape_dart

  Perform (execute) the the_behaviour::escape_dart behaviour. See `the_behaviour::behaviour_do_escape_dart()`.
- procedure, public do_approach => behaviour_do_approach

  Approach a specific the_environment::spatial class target, i.e. execute the the_behaviour::approach behaviour. See `the_behaviour::behaviour_do_approach()`.
- procedure, public do_migrate => behaviour_do_migrate

  Perform (execute) the the_behaviour::migrate (migration) behaviour. See `the_behaviour::behaviour_do_migrate()`.
- procedure, public migrate_random => behaviour_try_migrate_random

  Perform a simplistic random migration. If the agent is within a specific distance to the target environment, it emigrates there with a specific fixed probability. See `the_behaviour::behaviour_try_migrate_random()`.
- procedure, public do_go_down => behaviour_do_go_down

  Perform (execute) the the_behaviour::go_down_depth (go down) behaviour. See `the_behaviour::behaviour_do_go_down()`.
- procedure, public do_go_up => behaviour_do_go_up

  Perform (execute) the the_behaviour::go_up_depth (go up) behaviour. See `the_behaviour::behaviour_do_go_up()`.
- procedure, public do_behave => behaviour_select_optimal

  Select and execute the optimal behaviour, i.e. the behaviour which minimizes the expected GOS arousal. See `the_behaviour::behaviour_select_optimal()`.

Public Attributes

- type(eat_food) eat

  *Parameters that set the parameters of the behaviours and their **expectancies** (perceived consequences).*
- type(reproduce) reproduce
- type(walk_random) walk_random
- type(freeze) freeze
- type(escape_dart) escape_dart
- type(approach) approach_spatial
- type(approach_conspec) approach_conspec
- type(migrate) migrate
- type(go_down_depth) depth_down
- type(go_up_depth) depth_up
- type(debug_base) debug_base
- character(len=label_length) behaviour_label

  Overall label of the behaviour being executed. It is used only for outputs.

Indicator and debugging variables.

A history stack of behaviours (labels) that have been executed.

- character(len=label_length), dimension(history_size_behaviours) history_behave
- integer n_eats_all_indicator

  An indicator showing the cumulative count of the_behaviour::eat_food attempts (notwithstanding successful or failures).
- integer n_eaten_indicator

  An indicator showing the cumulative count of the food items eaten.
- real(srp) mass_eaten_indicator

  An indicator showing the cumulative mass of the food items eaten.

9.8.1 Detailed Description

The behaviour of the agent is defined by the the_behaviour::behaviour class. This class defines the *behavioural repertoire* of the agent. Each of the components of the behavioural repertoire (behaviour object) is defined as a separate independent class with its own *self* parameter. However, the agent which performs the behaviour (the *actor agent*) is included as the first non-self parameter into the behaviour component methods.
For example, there is a behaviour component the_behaviour::eat_food that defines the feeding behaviour of the agent. The method that calculates the basic (and expected) outputs from this behaviour (i.e. "does" it) the_behaviour::eat_food::do_this() includes the actor agent as the first non-self (non-`this`) parameter. The same is true for all other methods of the the_behaviour::eat_food class: the_behaviour::eat_food::expectancies_calculate() and the_behaviour::execute().
Thus, the many individual classes define the behavioural repertoire:

- the_behaviour::eat_food;
- the_behaviour::reproduce;
- the_behaviour::walk_random;
- the_behaviour::freeze;
- the_behaviour::escape_dart;
- the_behaviour::approach;
- the_behaviour::approach_conspec;
- the_behaviour::migrate;
- the_behaviour::go_down_depth;
- the_behaviour::go_up_depth.

However the_behaviour::behaviour unites all these classes together and plugs them into the agent class hierarchy.
An overview of the behavioural repertoire is found here.
Definition at line 514 of file m_behav.f90.

9.8.2 Member Function/Subroutine Documentation

9.8.2.1 init_behaviour()

`procedure, public the_behaviour::behaviour::init_behaviour`
The subroutines contained define what the agent really does, i.e. implements the actual behavioural repertoire components.
Initialise the behaviour components of the agent, the the_behaviour::behaviour class. See `the_behaviour::behaviour_whol`
Definition at line 551 of file m_behav.f90.

9.8.2.2 cleanup_behav_history()

`procedure, public the_behaviour::behaviour::cleanup_behav_history`
Cleanup the behaviour history stack for the agent. See `the_behaviour::behaviour_cleanup_history()`.
Definition at line 554 of file m_behav.f90.

9.8.2.3 deactivate()

`procedure, public the_behaviour::behaviour::deactivate`
Deactivate all behaviour units that compose the behaviour repertoire of the agent. See `the_behaviour::behaviour_whole_`
Definition at line 557 of file m_behav.f90.

9.8.2.4 behaviour_is()

procedure, public the_behaviour::behaviour::behaviour_is
Obtain the label of the currently executing behaviour for the this agent. See the_behaviour::behaviour_get_behaviou
Definition at line 560 of file m_behav.f90.

9.8.2.5 food_item_select()

procedure, public the_behaviour::behaviour::food_item_select
Select the optimal food item among (possibly) several ones that are available in the **perception object** of the agent. See the_behaviour::behaviour_select_food_item().
Definition at line 565 of file m_behav.f90.

9.8.2.6 consp_select()

procedure, public the_behaviour::behaviour::consp_select
Select the optimal conspecific among (possibly) several ones that are available in the **perception object** of the agent. See the_behaviour::behaviour_select_conspecific().
Definition at line 569 of file m_behav.f90.

9.8.2.7 food_item_select_nearest()

procedure, public the_behaviour::behaviour::food_item_select_nearest
Select the nearest food item among (possibly) several ones that are available in the perception object. See the_behaviour::behaviour_select_food_item_nearest().
Definition at line 573 of file m_behav.f90.

9.8.2.8 consp_select_nearest()

procedure, public the_behaviour::behaviour::consp_select_nearest
Select the nearest conspecific among (possibly) several ones that are available in the perception object. See the_behaviour::behaviour_select_conspecific_nearest().
Definition at line 578 of file m_behav.f90.

9.8.2.9 do_eat_food_item()

procedure, public the_behaviour::behaviour::do_eat_food_item
Eat a food item(s) that are found in the perception object. See the_behaviour::behaviour_do_eat_food_item().
Definition at line 583 of file m_behav.f90.

9.8.2.10 do_reproduce()

procedure, public the_behaviour::behaviour::do_reproduce
Reproduce based on the this agent's current state. See the_behaviour::behaviour_do_reproduce().
Definition at line 586 of file m_behav.f90.

9.8.2.11 do_walk()

procedure, public the_behaviour::behaviour::do_walk
Perform a random Gaussian walk to a specific average distance with certain variance (set by the CV). See the_behaviour::behaviour_do_walk().
Definition at line 590 of file m_behav.f90.

9.8.2.12 do_freeze()

procedure, public the_behaviour::behaviour::do_freeze
Perform (execute) the the_behaviour::freeze behaviour. See the_behaviour::behaviour_do_freeze().
Definition at line 593 of file m_behav.f90.

9.8.2.13 do_escape()

procedure, public the_behaviour::behaviour::do_escape
Perform (execute) the the_behaviour::escape_dart behaviour. See the_behaviour::behaviour_do_escape_dart().
Definition at line 596 of file m_behav.f90.

9.8.2.14 do_approach()

procedure, public the_behaviour::behaviour::do_approach
Approach a specific the_environment::spatial class target, i.e. execute the the_behaviour::approach behaviour. See the_behaviour::behaviour_do_approach().
Definition at line 600 of file m_behav.f90.

9.8.2.15 do_migrate()

procedure, public the_behaviour::behaviour::do_migrate
Perform (execute) the the_behaviour::migrate (migration) behaviour. See the_behaviour::behaviour_do_migrate().
Definition at line 603 of file m_behav.f90.

9.8.2.16 migrate_random()

procedure, public the_behaviour::behaviour::migrate_random
Perform a simplistic random migration. If the agent is within a specific distance to the target environment, it emigrates there with a specific fixed probability. See the_behaviour::behaviour_try_migrate_random().
Definition at line 608 of file m_behav.f90.

9.8.2.17 do_go_down()

procedure, public the_behaviour::behaviour::do_go_down
Perform (execute) the the_behaviour::go_down_depth (go down) behaviour. See the_behaviour::behaviour_do_go_down
Definition at line 611 of file m_behav.f90.

9.8.2.18 do_go_up()

procedure, public the_behaviour::behaviour::do_go_up
Perform (execute) the the_behaviour::go_up_depth (go up) behaviour. See the_behaviour::behaviour_do_go_up().
Definition at line 614 of file m_behav.f90.

9.8.2.19 do_behave()

procedure, public the_behaviour::behaviour::do_behave
Select and execute the optimal behaviour, i.e. the behaviour which minimizes the expected GOS arousal. See the_behaviour::behaviour_select_optimal().

- There is a different behaviour selection backend procedure that depends on the current GOS↩ : the_behaviour::behaviour_select_fixed_from_gos(). This procedure selects a specific fixed behaviour unit at specific GOS.

Definition at line 623 of file m_behav.f90.

9.8.3 Member Data Documentation

9.8.3.1 eat

type(eat_food) the_behaviour::behaviour::eat
Parameters that set the parameters of the behaviours and their **expectancies** (perceived consequences).
Definition at line 517 of file m_behav.f90.

9.8.3.2 reproduce

type(reproduce) the_behaviour::behaviour::reproduce
Definition at line 518 of file m_behav.f90.

9.8.3.3 walk_random

type(walk_random) the_behaviour::behaviour::walk_random
Definition at line 519 of file m_behav.f90.

9.8.3.4 freeze

type(freeze) the_behaviour::behaviour::freeze
Definition at line 520 of file m_behav.f90.

9.8.3.5 escape_dart

type(escape_dart) the_behaviour::behaviour::escape_dart
Definition at line 521 of file m_behav.f90.

9.8.3.6 approach_spatial

type(approach) the_behaviour::behaviour::approach_spatial
Definition at line 522 of file m_behav.f90.

9.8.3.7 approach_conspec

type(approach_conspec) the_behaviour::behaviour::approach_conspec
Definition at line 523 of file m_behav.f90.

9.8.3.8 migrate

type(migrate) the_behaviour::behaviour::migrate
Definition at line 524 of file m_behav.f90.

9.8.3.9 depth_down

type(go_down_depth) the_behaviour::behaviour::depth_down
Definition at line 525 of file m_behav.f90.

9.8.3.10 depth_up

`type(go_up_depth) the_behaviour::behaviour::depth_up`
Definition at line 526 of file m_behav.f90.

9.8.3.11 debug_base

`type(debug_base) the_behaviour::behaviour::debug_base`
Definition at line 527 of file m_behav.f90.

9.8.3.12 behaviour_label

`character(len=label_length) the_behaviour::behaviour::behaviour_label`
Overall label of the behaviour being executed. It is used only for outputs.
Definition at line 530 of file m_behav.f90.

9.8.3.13 history_behave

`character(len=label_length), dimension(history_size_behaviours) the_behaviour::behaviour↩ ::history_behave`
Definition at line 535 of file m_behav.f90.

9.8.3.14 n_eats_all_indicator

`integer the_behaviour::behaviour::n_eats_all_indicator`
An indicator showing the cumulative count of the_behaviour::eat_food attempts (notwithstanding successful or failures).
Definition at line 539 of file m_behav.f90.

9.8.3.15 n_eaten_indicator

`integer the_behaviour::behaviour::n_eaten_indicator`
An indicator showing the cumulative count of the food items eaten.
Definition at line 541 of file m_behav.f90.

9.8.3.16 mass_eaten_indicator

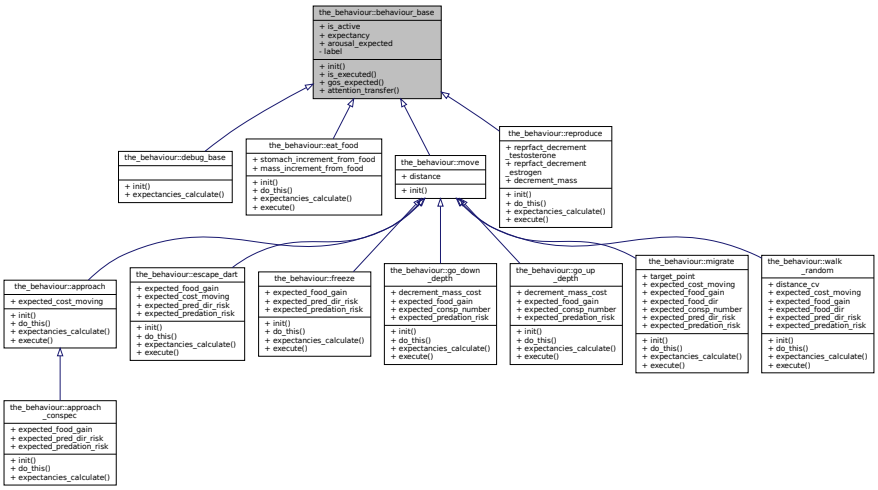
`real(srp) the_behaviour::behaviour::mass_eaten_indicator`
An indicator showing the cumulative mass of the food items eaten.
Definition at line 543 of file m_behav.f90.
The documentation for this type was generated from the following file:

- m_behav.f90

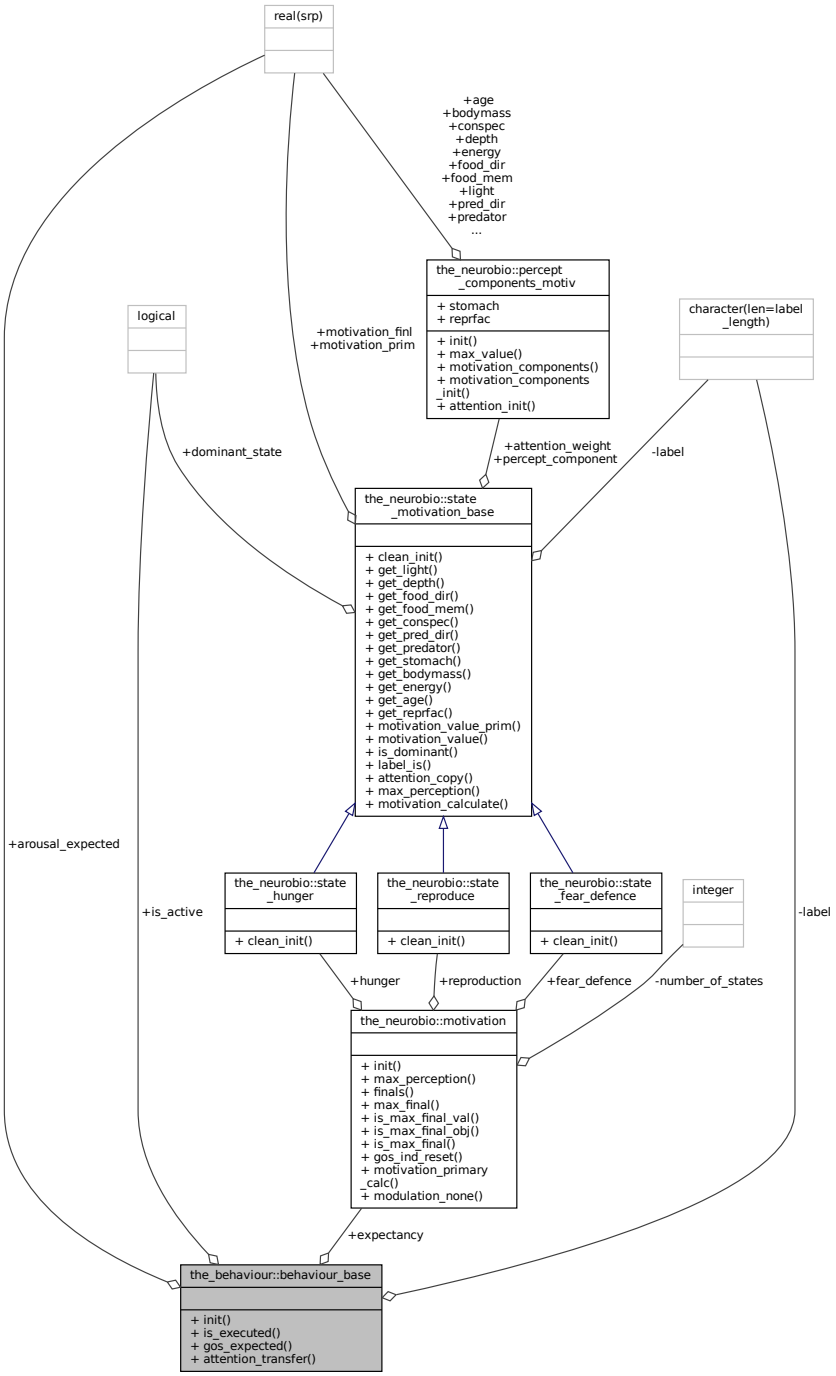
9.9 the_behaviour::behaviour_base Type Reference

Root behaviour abstract type. Several different discrete behaviours encompass the behavioural repertoire of the agent. This is the base root type from which all other behaviours are obtained by inheritance/extension.

Inheritance diagram for the_behaviour::behaviour_base:



Collaboration diagram for the_behaviour::behaviour_base:



Public Member Functions

- procedure(behaviour_init_root), deferred, public init

  *Abstract **init** function that has to be overriden by each object that extends the root behaviour component class.*
- procedure, public is_executed => behaviour_root_get_is_executed

  Get the execution status of the behaviour unit. See `the_behaviour::behaviour_root_get_is_executed()`.
- procedure, public gos_expected => behaviour_root_gos_expectation

gos_expected is an accessor get-function that returns the final GOS expectation from expectancies_↩ calculate. Once we get this value for all the possible behaviours, we choose what behaviour to execute by minimising gos_expected. See the_behaviour::behaviour_root_gos_expectation().

- procedure, public attention_transfer => behaviour_root_attention_weights_transfer

  attention_transfer transfers attention weights from the actor agent to this behaviour expectancy objects. See the_behaviour::behaviour_root_attention_weights_transfer().

Public Attributes

- logical is_active

  Logical flag indicating that this behaviour is activated (executed).
- type(motivation) expectancy

  *Each behavioural type within the whole repertoire has **expectancies** that set how each of the GOS motivational components is affected by its execution.*
- real(srp) arousal_expected

  An expectation of the arousal level. It is the maximum weighted value among all motivation components. This value is actually minimised – those behaviour which would result in the lowest arousal_expected is finally executed.

Private Attributes

- character(len=label_length), private label

  Label for the behaviour type.

9.9.1 Detailed Description

Root behaviour abstract type. Several different discrete behaviours encompass the behavioural repertoire of the agent. This is the base root type from which all other behaviours are obtained by inheritance/extension.
Definition at line 36 of file m_behav.f90.

9.9.2 Member Function/Subroutine Documentation

9.9.2.1 init()

procedure(behaviour_init_root), deferred, public the_behaviour::behaviour_base::init
Abstract **init** function that has to be overriden by each object that extends the root behaviour component class.
Definition at line 54 of file m_behav.f90.

9.9.2.2 is_executed()

procedure, public the_behaviour::behaviour_base::is_executed
Get the execution status of the behaviour unit. See the_behaviour::behaviour_root_get_is_executed().
Definition at line 57 of file m_behav.f90.

9.9.2.3 gos_expected()

procedure, public the_behaviour::behaviour_base::gos_expected
gos_expected is an accessor get-function that returns the final GOS expectation from expectancies_↩ calculate. Once we get this value for all the possible behaviours, we choose what behaviour to execute by minimising gos_expected. See the_behaviour::behaviour_root_gos_expectation().
Definition at line 63 of file m_behav.f90.

9.9.2.4 attention_transfer()

`procedure, public the_behaviour::behaviour_base::attention_transfer`
`attention_transfer` transfers attention weights from the actor agent to this behaviour expectancy objects. See `the_behaviour::behaviour_root_attention_weights_transfer()`.
Definition at line 67 of file m_behav.f90.

9.9.3 Member Data Documentation

9.9.3.1 label

`character(len=label_length), private the_behaviour::behaviour_base::label [private]`
Label for the behaviour type.
Definition at line 38 of file m_behav.f90.

9.9.3.2 is_active

`logical the_behaviour::behaviour_base::is_active`
Logical flag indicating that this behaviour is activated (executed).

Warning

Only one behaviour unit can be executed at a time.

Definition at line 41 of file m_behav.f90.

9.9.3.3 expectancy

`type(motivation) the_behaviour::behaviour_base::expectancy`
Each behavioural type within the whole repertoire has **expectancies** that set how each of the GOS motivational components is affected by its execution.
Definition at line 45 of file m_behav.f90.

9.9.3.4 arousal_expected

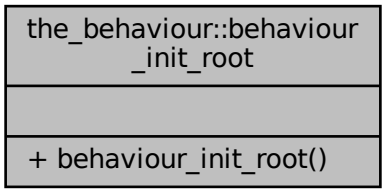
`real(srp) the_behaviour::behaviour_base::arousal_expected`
An expectation of the arousal level. It is the maximum weighted value among all motivation components. This value is actually minimised – those behaviour which would result in the lowest `arousal_expected` is finally executed.
Definition at line 50 of file m_behav.f90.
The documentation for this type was generated from the following file:

- m_behav.f90

9.10 the_behaviour::behaviour_init_root Interface Reference

Abstract interface for the deferred **init** function that has to be overriden by each object that extends the basic behavioural component class.

Collaboration diagram for the_behaviour::behaviour_init_root:



Public Member Functions

- elemental subroutine behaviour_init_root (this)

9.10.1 Detailed Description

Abstract interface for the deferred **init** function that has to be overriden by each object that extends the basic behavioural component class.
Definition at line 76 of file m_behav.f90.

9.10.2 Constructor & Destructor Documentation

9.10.2.1 behaviour_init_root()

```
elemental subroutine the_behaviour::behaviour_init_root::behaviour_init_root (
            class(behaviour_base), intent(inout) this )
```

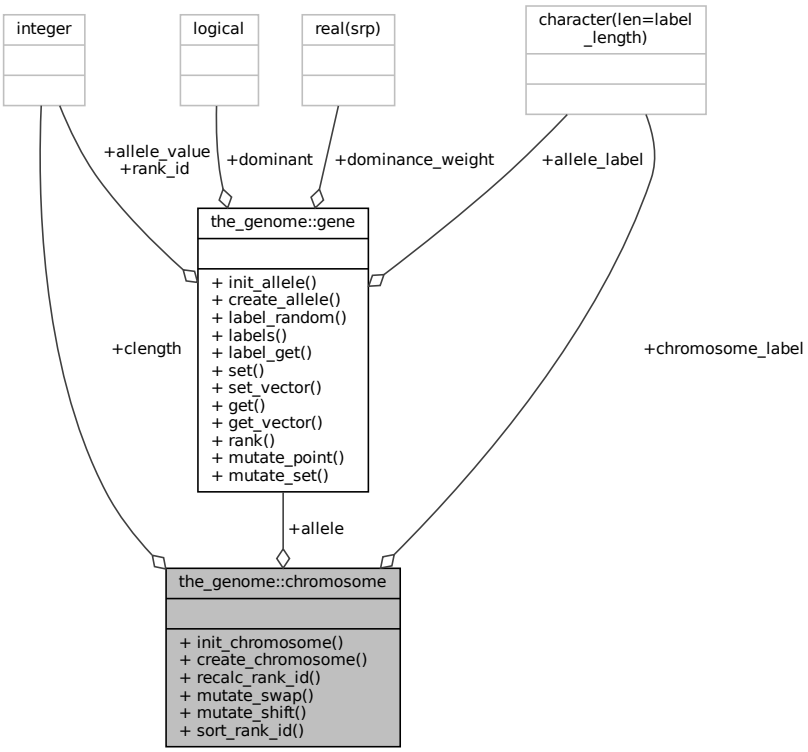
Definition at line 76 of file m_behav.f90.
The documentation for this interface was generated from the following file:

- m_behav.f90

9.11 the_genome::chromosome Type Reference

This type describes the chromosome object. Chromosome consists of an array of alleles and a descriptive string label. See "the genome structure" for as general description and "chromosome" for details.

Collaboration diagram for the_genome::chromosome:



Public Member Functions

- procedure, public init_chromosome => chromosome_init_allocate_random

  This subroutine initialises the chromosome with, and allocates, random alleles, sets one of them randomly dominant and optionally defines the chromosome label. See `the_genome::chromosome_init_allocate_random()`

- procedure, public create_chromosome => chromosome_create_allocate_zero

  Init a new chromosome, zero, non-random. See `the_genome::chromosome_create_allocate_zero()`

- procedure, public recalc_rank_id => chromosome_recalculate_rank_ids

  This subroutine recalculates rank_id indices for consecutive gene objects within the chromosome. This may be necessary after reordering by random relocation mutation. See `the_genome::chromosome_recalculate_rank_ids()`

- procedure, public mutate_swap => chromosome_mutate_relocate_swap_random

  mutate within the same chromosome, relocate a gene (unit of alleles) to a different random position within the same chromosome, the misplaced gene moves to the relocated gene position, so they are just swap. See `the_genome::chromosome_mutate_relocate_swap_random()`

- procedure, public mutate_shift => chromosome_mutate_relocate_shift_random

  Mutate within the same chromosome, relocate a gene (unit of alleles) to a different random position within the same chromosome, shifting all other genes within the chromosome down one position. This works as follows: first, we randomly determine the gene to relocate, assign it a new random rank_id. Then re-sort the chromosome according to the new ranks with `qsort` *with the* `qs_partition_rank_id` *backend. See* `the_genome::chromosome_mutate_relocate_shift_random()`

- procedure, public sort_rank_id => chromosome_sort_rank_id

  Sort GENE objects within the CHROMOSOME by their rank_id The two subroutines below are a variant of the recursive quick-sort algorithm adapted for sorting integer components of the the `CHROMOSOME` *object. See* `the_genome::chromosome_sort_rank_id()`

Public Attributes

- character(len=label_length) chromosome_label

  chromosome label
- integer clength

  chromosome length, i.e. N of alleles here
- type(gene), dimension(:), allocatable allele

  array of alleles of the size `clength`

9.11.1 Detailed Description

This type describes the chromosome object. Chromosome consists of an array of alleles and a descriptive string label. See "the genome structure" for as general description and "chromosome" for details.
Definition at line 110 of file m_genome.f90.

9.11.2 Member Function/Subroutine Documentation

9.11.2.1 init_chromosome()

`procedure, public the_genome::chromosome::init_chromosome`
This subroutine initialises the chromosome with, and allocates, random alleles, sets one of them randomly dominant and optionally defines the chromosome label. See `the_genome::chromosome_init_allocate_random()`
Definition at line 122 of file m_genome.f90.

9.11.2.2 create_chromosome()

`procedure, public the_genome::chromosome::create_chromosome`
Init a new chromosome, zero, non-random. See `the_genome::chromosome_create_allocate_zero()`
Definition at line 125 of file m_genome.f90.

9.11.2.3 recalc_rank_id()

`procedure, public the_genome::chromosome::recalc_rank_id`
This subroutine recalculates rank_id indices for consecutive gene objects within the chromosome. This may be necessary after reordering by random relocation mutation. See `the_genome::chromosome_recalculate_rank_ids()`
Definition at line 130 of file m_genome.f90.

9.11.2.4 mutate_swap()

`procedure, public the_genome::chromosome::mutate_swap`
mutate within the same chromosome, relocate a gene (unit of alleles) to a different random position within the same chromosome, the misplaced gene moves to the relocated gene position, so they are just swap. See `the_genome::chromosome_mutate_relocate_swap_random()`
Definition at line 135 of file m_genome.f90.

9.11.2.5 mutate_shift()

`procedure, public the_genome::chromosome::mutate_shift`
Mutate within the same chromosome, relocate a gene (unit of alleles) to a different random position within the same chromosome, shifting all other genes within the chromosome down one position. This works as follows: first, we randomly determine the gene to relocate, assign it a new random rank_id. Then re-sort the chromosome according to the new ranks with `qsort` with the `qs_partition_rank_id` backend. See `the_genome::chromosome_mutate_relocate_shift_random()`

Definition at line 143 of file m_genome.f90.

9.11.2.6 sort_rank_id()

`procedure, public the_genome::chromosome::sort_rank_id`
Sort GENE objects within the CHROMOSOME by their rank_id The two subroutines below are a variant of the recursive quick-sort algorithm adapted for sorting integer components of the the `CHROMOSOME` object. See `the_genome::chromosome_sort_rank_id()`
Definition at line 149 of file m_genome.f90.

9.11.3 Member Data Documentation

9.11.3.1 chromosome_label

`character(len=label_length) the_genome::chromosome::chromosome_label`
chromosome label
Definition at line 112 of file m_genome.f90.

9.11.3.2 clength

`integer the_genome::chromosome::clength`
chromosome length, i.e. N of alleles here
Definition at line 114 of file m_genome.f90.

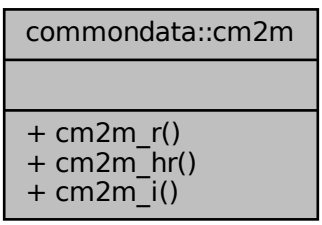
9.11.3.3 allele

`type(gene), dimension(:), allocatable the_genome::chromosome::allele`
array of alleles of the size `clength`
Definition at line 116 of file m_genome.f90.
The documentation for this type was generated from the following file:

- m_genome.f90

9.12 commondata::cm2m Interface Reference

Convert cm to m.
Collaboration diagram for commondata::cm2m:



Public Member Functions

- elemental real(srp) function cm2m_r (value_cm)

  Convert cm to m.
- elemental real(hrp) function cm2m_hr (value_cm)

  Convert cm to m.
- elemental real(srp) function cm2m_i (value_cm)

  Convert cm to m.

9.12.1 Detailed Description

Convert cm to m.
This is needed because some of the the sizes are expressed in cm but certain functions (e.g. visual range estimator SRGETR) require parameters in m. E.g. FOOD_ITEM_SIZE_DEFAULT is set around 0.5 cm while SRGETR requires prey area in m^2.
Definition at line 5306 of file m_common.f90.

9.12.2 Member Function/Subroutine Documentation

9.12.2.1 cm2m_r()

```
elemental real(srp) function commondata::cm2m::cm2m_r (
            real(srp), intent(in) value_cm )
```

Convert cm to m.

Parameters

value_cm	value in cm

Returns

value in m

Note

This version gets real type argument.

This is needed because some of the the sizes are expressed in cm but certain functions (e.g. visual range estimator SRGETR) require parameters in m. E.g. FOOD_ITEM_SIZE_DEFAULT is set around 0.5 cm while SRGETR requires prey area in m^2.
Definition at line 5550 of file m_common.f90.

9.12.2.2 cm2m_hr()

```
elemental real(hrp) function commondata::cm2m::cm2m_hr (
            real(hrp), intent(in) value_cm )
```

Convert cm to m.

Parameters

value_cm	value in cm

Returns

value in m

Note

This version uses the **high** HRP numerical precision model.

Definition at line 5564 of file m_common.f90.

9.12.2.3 cm2m_i()

```
elemental real(srp) function commondata::cm2m::cm2m_i (
            integer, intent(in) value_cm )
```

Convert cm to m.

Returns

value in m

Parameters

value_cm	value in cm

Note

This version gets integer argument (albeit returns real).

Definition at line 5578 of file m_common.f90.
The documentation for this interface was generated from the following file:

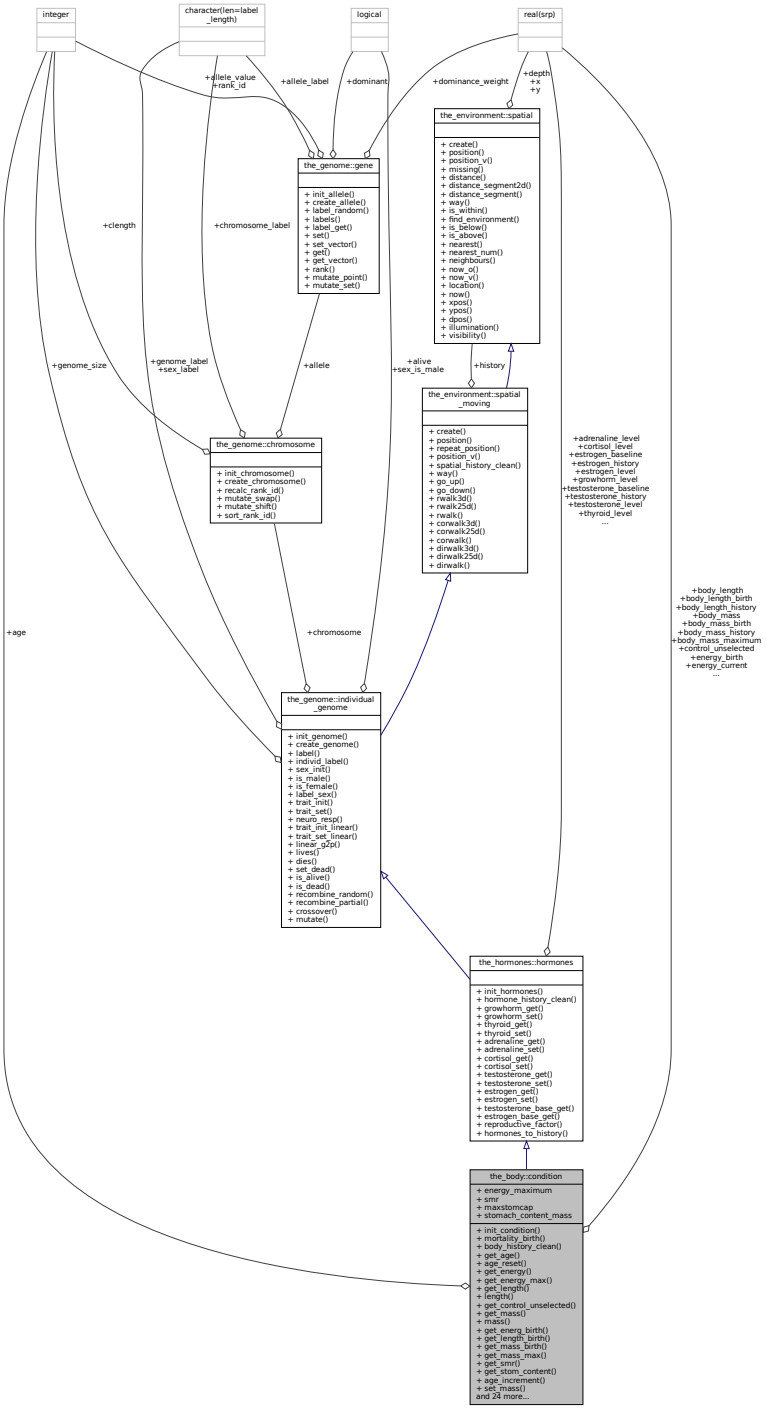
- m_common.f90

9.13 the_body::condition Type Reference

CONDITION defines the physical condition of the agent

Inheritance diagram for the_body::condition:

Collaboration diagram for the_body::condition:



Public Member Functions

- procedure, public init_condition => condition_init_genotype

  Initialise the individual body condition object based on the genome values. See `the_body::condition_init_genotype()`
- procedure, public mortality_birth => birth_mortality_enforce_init_fixed_debug

  This procedure enforces selective mortality of agents at birth. See `the_body::birth_mortality_enforce_init_fixed_de`
- procedure, public body_history_clean => condition_clean_history

  Cleanup the history stack of the body length and mass. See `the_body::condition_clean_history()`

- procedure, public get_age => condition_age_get

  Get current age. See `the_body::condition_age_get()`
- procedure, public age_reset => condition_age_reset_zero

  Reset the age of the agent to zero. See `the_body::condition_age_reset_zero()`.
- procedure, public get_energy => condition_energy_current_get

  Get current energy reserves. See `the_body::condition_energy_current_get()`
- procedure, public get_energy_max => condition_energy_maximum_get

  Get historical maximum of energy reserves. See `the_body::condition_energy_maximum_get()`
- procedure, public get_length => condition_body_length_get

  Get current body length. See `the_body::condition_body_length_get()`
- generic, public length => get_length

  Generic interface (alias) for `get_length`.
- procedure, public get_control_unselected => condition_control_unsel_get

  Get current value of the control unselected trait. See `the_body:condition_control_unsel_get:()`
- procedure, public get_mass => condition_body_mass_get

  Get current body mass. See `the_body::condition_body_mass_get()`
- generic, public mass => get_mass

  Generic interface to get_mass.
- procedure, public get_energ_birth => condition_energy_birth_get

  Get historical record of energy reserves at birth. See `the_body::condition_energy_birth_get()`.
- procedure, public get_length_birth => condition_body_length_birth_get

  Get historical record of body length at birth. See `the_body::condition_body_length_birth_get()`
- procedure, public get_mass_birth => condition_body_mass_birth_get

  Get historical record of body mass at birth. See `the_body::condition_body_mass_birth_get()`
- procedure, public get_mass_max => condition_body_mass_max_get

  Get historcal maximum for body mass. See `the_body::condition_body_mass_max_get()`
- procedure, public get_smr => condition_smr_get

  Get current smr. See `the_body::condition_smr_get()`
- procedure, public get_stom_content => condition_stomach_content_get

  Get current stomach content. See `the_body::condition_stomach_content_get()`
- procedure, public age_increment => condition_age_increment

  Increment the age of the agent by one. See `the_body::condition_age_increment()`.
- procedure, public set_mass => condition_body_mass_set_update_hist

  Set body mass optionally updating the history stack. See `the_body::condition_body_mass_set_update_hist()`
- procedure, public set_length => condition_body_length_set_update_hist

  Set body length optionally updating the history stack. See `the_body::condition_body_length_set_update_hist()`
- procedure, public visibility => condition_agent_visibility_visual_range

  Calculate the visibility range of this agent. Visibility depends on the size of the agent, ambient illumination and agent contrast. Visibility is the distance from which this agent can be seen by a visual object (e.g. predator or conspecific). See `the_body::condition_agent_visibility_visual_range`.
- procedure, public food_proc_cost_v => body_mass_processing_cost_calc_v

  Calculate the basic processing cost of catching a food item with the mass `food_gain`. *Vector-based procedure. See* `the_body::body_mass_processing_cost_calc_v()`
- procedure, public food_proc_cost_o => body_mass_processing_cost_calc_o

  Calculate the basic processing cost of catching a food item with the mass `food_gain`. *Object-based procedure. See* `the_body::body_mass_processing_cost_calc_o()`
- generic, public food_process_cost => food_proc_cost_v, food_proc_cost_o

  Generic interface to the procedures calculating the basic processing cost of catching a food item.
- procedure, public food_fitt_v => stomach_content_food_gain_fitting_v

  Calculate the value of possible food gain as fitting into the agent's stomach (or full gain if the food item fits wholly). Vector-based. See `the_body::stomach_content_food_gain_fitting_v()`
- procedure, public food_fitt_o => stomach_content_food_gain_fitting_o

Calculate the value of possible food gain as fitting into the agent's stomach (or full gain if the food item fits wholly). Object-based. See `the_body::stomach_content_food_gain_fitting_o()`

- generic, public food_fitting => food_fitt_v, food_fitt_o

  Generic interface to procedures that calculate the value of possible food gain as fitting into the agent's stomach (or full gain if the food item fits wholly). See `the_body::stomach_content_food_gain_fitting_v()` *and* `the_body::stomach_content_food_gain_fitting_o()`.
- procedure, public food_surpl_v => stomach_content_food_gain_non_fit_v

  Calculate extra food surplus mass non fitting into the stomach of the agent. Vector-based. See `the_body::stomach_content_foo`
- procedure, public food_surpl_o => stomach_content_food_gain_non_fit_o

  Calculate extra food surplus mass non fitting into the stomach of the agent. Object-based. See `the_body::stomach_content_foo`
- generic, public food_surplus => food_surpl_v, food_surpl_o

  Generic interface to procedures that calculate extra food surplus mass non fitting into the stomach of the agent.
- procedure, public mass_grow => body_mass_grow_do_calculate

  Do grow body mass based on food gain from a single food item adjusted for cost etc. See `the_body::body_mass_grow_do_calc`
- procedure, public stomach_increment => stomach_content_get_increment

  Do increment stomach contents with adjusted (fitted) value. See `the_body::stomach_content_get_increment()`
- procedure, public cost_factor_food_smr => body_mass_food_processing_cost_factor_smr

  The fraction of the cost of the processing of the food item(s) depending on the agent SMR. It is scaled in terms of the ratio of the food item mass to the agent mass. See `the_body::body_mass_food_processing_cost_factor_smr()`
- procedure, public cost_swim => condition_cost_swimming_burst

  The cost of swimming of a specific distance in terms of body mass loss. See `the_body::condition_cost_swimming_burst()`
- procedure, public cost_swim_std => cost_swimming_standard

  The standard cost of swimming is a diagnostic function that shows the cost, in units of the body mass, incurred if the agent passes a distance equal to commondata::lifespan units of its body length. See `the_body::cost_swimming_standard()`.
- procedure, public energy_update => condition_energy_update_after_growth

  Update the energy reserves of the agent based on its current mass and length. See `the_body::condition_energy_update_af`
- procedure, public starved_death => body_mass_is_starvation_check

  Check if the body mass is smaller than the birth body mass or structural body mass. See `the_body::body_mass_is_starvation`
- procedure, public living_cost => body_mass_calculate_cost_living_step

  Calculate the cost of living for a single model step. See `the_body::body_mass_calculate_cost_living_step()`
- procedure, public subtract_living_cost => body_mass_adjust_living_cost_step

  Adjust the body mass at the end of the model step against the cost of living. See `the_body::body_mass_adjust_living_cost`
- procedure, public len_incr => body_len_grow_calculate_increment_step

  Calculate body length increment. See `the_body::body_len_grow_calculate_increment_step()`
- procedure, public len_grow => body_len_grow_do_calculate_step

  Do linear growth for one model step based on the above increment function. See `the_body::body_len_grow_do_calculate_s`
- procedure, public stomach_empify => stomach_content_mass_emptify_step

  Do digestion. Stomach contents S(t) is emptied by a constant fraction each time step. See details in the `stomach_content_mass_emptify_step` *function call. See* `the_body::stomach_content_mass_emptify_step()`
- procedure, public sex_steroids_update => sex_steroids_update_increment

  Update the level of the sex steroids. Sex steroids are incremented each time step of the model. See `the_body::sex_steroids_update_increment()`

Public Attributes

- integer age

  The age of the agent in units of the integer time steps.
- real(srp) energy_current

  Current energy reserves, initialised non-genetically, Gaussian.
- real(srp) energy_maximum

  Maximum historical energy reserves.
- real(srp) energy_birth

Energy reserves at birth, non-genetic, Gaussian.

- real(srp) body_length

  current body length, initialised non-genetically, Gaussian, will grow.
- real(srp), dimension(history_size_agent_prop) body_length_history

  History stack for the body length.
- real(srp) body_length_birth

  Body length at birth (genetically fixed), it is not used so far in the calculations but is recorded and can be output.
- real(srp) control_unselected

  *This is a **control unselected** (and unused) trait that is set from the genome as normal but is not used in any calculations. It can be used as a control marker for random genetic drift.*
- real(srp) body_mass

  Current body mass, initialised calculated from length and energy reserves.
- real(srp), dimension(history_size_agent_prop) body_mass_history

  History stack for body mass.
- real(srp) body_mass_birth

  Body mass at birth, will keep record of it.
- real(srp) body_mass_maximum

  Maximum historically body, will keep record of it.
- real(srp) smr

  Standard metabolitic rate, can change depending on hormones and psychological state (GOS)), at birth initialised from the genome.
- real(srp) maxstomcap = MAX_STOMACH_CAPACITY_DEF

  Maximum stomach capacity, max. fraction of body mass available for food here set from default value. But can change in future versions of the model depending on the body length and the physiological state (so specifically set for each agent rather than defined from global parameter commondata::max_stomach_capacity_def).
- real(srp) stomach_content_mass

  Stomach content mass.

9.13.1 Detailed Description

CONDITION defines the physical condition of the agent
Definition at line 29 of file m_body.f90.

9.13.2 Member Function/Subroutine Documentation

9.13.2.1 init_condition()

procedure, public the_body::condition::init_condition
Initialise the individual body condition object based on the genome values. See the_body::condition_init_genotype()
Definition at line 79 of file m_body.f90.

9.13.2.2 mortality_birth()

procedure, public the_body::condition::mortality_birth
This procedure enforces selective mortality of agents at birth. See the_body::birth_mortality_enforce_init_fixed_
Definition at line 82 of file m_body.f90.

9.13.2.3 body_history_clean()

procedure, public the_body::condition::body_history_clean
Cleanup the history stack of the body length and mass. See the_body::condition_clean_history()
Definition at line 86 of file m_body.f90.

9.13.2.4 get_age()

procedure, public the_body::condition::get_age
Get current age. See the_body::condition_age_get()
Definition at line 90 of file m_body.f90.

9.13.2.5 age_reset()

procedure, public the_body::condition::age_reset
Reset the age of the agent to zero. See the_body::condition_age_reset_zero().
Definition at line 93 of file m_body.f90.

9.13.2.6 get_energy()

procedure, public the_body::condition::get_energy
Get current energy reserves. See the_body::condition_energy_current_get()
Definition at line 96 of file m_body.f90.

9.13.2.7 get_energy_max()

procedure, public the_body::condition::get_energy_max
Get historical maximum of energy reserves. See the_body::condition_energy_maximum_get()
Definition at line 99 of file m_body.f90.

9.13.2.8 get_length()

procedure, public the_body::condition::get_length
Get current body length. See the_body::condition_body_length_get()
Definition at line 102 of file m_body.f90.

9.13.2.9 length()

generic, public the_body::condition::length
Generic interface (alias) for get_length.
Definition at line 104 of file m_body.f90.

9.13.2.10 get_control_unselected()

procedure, public the_body::condition::get_control_unselected
Get current value of the control unselected trait. See the_body:condition_control_unsel_get:()
Definition at line 107 of file m_body.f90.

9.13.2.11 get_mass()

procedure, public the_body::condition::get_mass
Get current body mass. See the_body::condition_body_mass_get()
Definition at line 110 of file m_body.f90.

9.13.2.12 mass()

generic, public the_body::condition::mass
Generic interface to get_mass.

Definition at line 112 of file m_body.f90.

9.13.2.13 get_energ_birth()

`procedure, public the_body::condition::get_energ_birth`
Get historical record of energy reserves at birth. See the_body::condition_energy_birth_get().
Definition at line 115 of file m_body.f90.

9.13.2.14 get_length_birth()

`procedure, public the_body::condition::get_length_birth`
Get historical record of body length at birth. See the_body::condition_body_length_birth_get()
Definition at line 118 of file m_body.f90.

9.13.2.15 get_mass_birth()

`procedure, public the_body::condition::get_mass_birth`
Get historical record of body mass at birth. See the_body::condition_body_mass_birth_get()
Definition at line 121 of file m_body.f90.

9.13.2.16 get_mass_max()

`procedure, public the_body::condition::get_mass_max`
Get historcal maximum for body mass. See the_body::condition_body_mass_max_get()
Definition at line 124 of file m_body.f90.

9.13.2.17 get_smr()

`procedure, public the_body::condition::get_smr`
Get current smr. See the_body::condition_smr_get()
Definition at line 127 of file m_body.f90.

9.13.2.18 get_stom_content()

`procedure, public the_body::condition::get_stom_content`
Get current stomach content. See the_body::condition_stomach_content_get()
Definition at line 130 of file m_body.f90.

9.13.2.19 age_increment()

`procedure, public the_body::condition::age_increment`
Increment the age of the agent by one. See the_body::condition_age_increment().
Definition at line 134 of file m_body.f90.

9.13.2.20 set_mass()

`procedure, public the_body::condition::set_mass`
Set body mass optionally updating the history stack. See the_body::condition_body_mass_set_update_hist()
Definition at line 137 of file m_body.f90.

9.13.2.21 set_length()

procedure, public the_body::condition::set_length
Set body length optionally updating the history stack. See the_body::condition_body_length_set_update_hist()
Definition at line 140 of file m_body.f90.

9.13.2.22 visibility()

procedure, public the_body::condition::visibility
Calculate the visibility range of this agent. Visibility depends on the size of the agent, ambient illumination and agent contrast. Visibility is the distance from which this agent can be seen by a visual object (e.g. predator or conspecific).
See the_body::condition_agent_visibility_visual_range.
Definition at line 146 of file m_body.f90.

9.13.2.23 food_proc_cost_v()

procedure, public the_body::condition::food_proc_cost_v
Calculate the basic processing cost of catching a food item with the mass food_gain. Vector-based procedure.
See the_body::body_mass_processing_cost_calc_v()
Definition at line 153 of file m_body.f90.

9.13.2.24 food_proc_cost_o()

procedure, public the_body::condition::food_proc_cost_o
Calculate the basic processing cost of catching a food item with the mass food_gain. Object-based procedure.
See the_body::body_mass_processing_cost_calc_o()
Definition at line 157 of file m_body.f90.

9.13.2.25 food_process_cost()

generic, public the_body::condition::food_process_cost
Generic interface to the procedures calculating the basic processing cost of catching a food item.
Definition at line 160 of file m_body.f90.

9.13.2.26 food_fitt_v()

procedure, public the_body::condition::food_fitt_v
Calculate the value of possible food gain as fitting into the agent's stomach (or full gain if the food item fits wholly). Vector-based. See the_body::stomach_content_food_gain_fitting_v()
Definition at line 166 of file m_body.f90.

9.13.2.27 food_fitt_o()

procedure, public the_body::condition::food_fitt_o
Calculate the value of possible food gain as fitting into the agent's stomach (or full gain if the food item fits wholly). Object-based. See the_body::stomach_content_food_gain_fitting_o()
Definition at line 170 of file m_body.f90.

9.13.2.28 food_fitting()

generic, public the_body::condition::food_fitting

Generic interface to procedures that calculate the value of possible food gain as fitting into the agent's stomach (or full gain if the food item fits wholly). See the_body::stomach_content_food_gain_fitting_v() and the_body::stomach_content_food_gain_fitting_o().
Definition at line 176 of file m_body.f90.

9.13.2.29 food_surpl_v()

procedure, public the_body::condition::food_surpl_v
Calculate extra food surplus mass non fitting into the stomach of the agent. Vector-based. See the_body::stomach_content_food_gain_non_fit_v()
Definition at line 181 of file m_body.f90.

9.13.2.30 food_surpl_o()

procedure, public the_body::condition::food_surpl_o
Calculate extra food surplus mass non fitting into the stomach of the agent. Object-based. See the_body::stomach_content_food_gain_non_fit_o()
Definition at line 185 of file m_body.f90.

9.13.2.31 food_surplus()

generic, public the_body::condition::food_surplus
Generic interface to procedures that calculate extra food surplus mass non fitting into the stomach of the agent.
Definition at line 188 of file m_body.f90.

9.13.2.32 mass_grow()

procedure, public the_body::condition::mass_grow
Do grow body mass based on food gain from a single food item adjusted for cost etc. See the_body::body_mass_grow_do_ca
Definition at line 193 of file m_body.f90.

9.13.2.33 stomach_increment()

procedure, public the_body::condition::stomach_increment
Do increment stomach contents with adjusted (fitted) value. See the_body::stomach_content_get_increment()
Definition at line 196 of file m_body.f90.

9.13.2.34 cost_factor_food_smr()

procedure, public the_body::condition::cost_factor_food_smr
The fraction of the cost of the processing of the food item(s) depending on the agent SMR. It is scaled in terms of the ratio of the food item mass to the agent mass. See the_body::body_mass_food_processing_cost_factor_smr()
Definition at line 202 of file m_body.f90.

9.13.2.35 cost_swim()

procedure, public the_body::condition::cost_swim
The cost of swimming of a specific distance in terms of body mass loss. See the_body::condition_cost_swimming_burs
Definition at line 206 of file m_body.f90.

9.13.2.36 cost_swim_std()

procedure, public the_body::condition::cost_swim_std
The standard cost of swimming is a diagnostic function that shows the cost, in units of the body mass, incurred if the agent passes a distance equal to commondata::lifespan units of its body length. See the_body::cost_swimming_standard().
Definition at line 211 of file m_body.f90.

9.13.2.37 energy_update()

procedure, public the_body::condition::energy_update
Update the energy reserves of the agent based on its current mass and length. See the_body::condition_energy_update
Definition at line 216 of file m_body.f90.

9.13.2.38 starved_death()

procedure, public the_body::condition::starved_death
Check if the body mass is smaller than the birth body mass or structural body mass. See the_body::body_mass_is_starvat
Definition at line 220 of file m_body.f90.

9.13.2.39 living_cost()

procedure, public the_body::condition::living_cost
Calculate the cost of living for a single model step. See the_body::body_mass_calculate_cost_living_step()
Definition at line 227 of file m_body.f90.

9.13.2.40 subtract_living_cost()

procedure, public the_body::condition::subtract_living_cost
Adjust the body mass at the end of the model step against the cost of living. See the_body::body_mass_adjust_living_co
Definition at line 231 of file m_body.f90.

9.13.2.41 len_incr()

procedure, public the_body::condition::len_incr
Calculate body length increment. See the_body::body_len_grow_calculate_increment_step()
Definition at line 234 of file m_body.f90.

9.13.2.42 len_grow()

procedure, public the_body::condition::len_grow
Do linear growth for one model step based on the above increment function. See the_body::body_len_grow_do_calculat
Definition at line 238 of file m_body.f90.

9.13.2.43 stomach_empify()

procedure, public the_body::condition::stomach_empify
Do digestion. Stomach contents S(t) is emptied by a constant fraction each time step. See details in the stomach↩
_content_mass_emptify_step function call. See the_body::stomach_content_mass_emptify_step()
Definition at line 243 of file m_body.f90.

9.13.2.44 sex_steroids_update()

`procedure, public the_body::condition::sex_steroids_update`
Update the level of the sex steroids. Sex steroids are incremented each time step of the model. See `the_body::sex_steroids_update_increment()`
Definition at line 247 of file m_body.f90.

9.13.3 Member Data Documentation

9.13.3.1 age

`integer the_body::condition::age`
The age of the agent in units of the integer time steps.
Definition at line 31 of file m_body.f90.

9.13.3.2 energy_current

`real(srp) the_body::condition::energy_current`
Current energy reserves, initialised non-genetically, Gaussian.
Definition at line 33 of file m_body.f90.

9.13.3.3 energy_maximum

`real(srp) the_body::condition::energy_maximum`
Maximum historical energy reserves.
Definition at line 35 of file m_body.f90.

9.13.3.4 energy_birth

`real(srp) the_body::condition::energy_birth`
Energy reserves at birth, non-genetic, Gaussian.
Definition at line 37 of file m_body.f90.

9.13.3.5 body_length

`real(srp) the_body::condition::body_length`
current body length, initialised non-genetically, Gaussian, will grow.
Definition at line 39 of file m_body.f90.

9.13.3.6 body_length_history

`real(srp), dimension(history_size_agent_prop) the_body::condition::body_length_history`
History stack for the body length.
Definition at line 41 of file m_body.f90.

9.13.3.7 body_length_birth

`real(srp) the_body::condition::body_length_birth`
Body length at birth (genetically fixed), it is not used so far in the calculations but is recorded and can be output.
Definition at line 44 of file m_body.f90.

9.13.3.8 control_unselected

`real(srp) the_body::condition::control_unselected`
This is a **control unselected** (and unused) trait that is set from the genome as normal but is not used in any calculations. It can be used as a control marker for random genetic drift.
Definition at line 48 of file m_body.f90.

9.13.3.9 body_mass

`real(srp) the_body::condition::body_mass`
Current body mass, initialised calculated from length and energy reserves.
Definition at line 51 of file m_body.f90.

9.13.3.10 body_mass_history

`real(srp), dimension(history_size_agent_prop) the_body::condition::body_mass_history`
History stack for body mass.
Definition at line 53 of file m_body.f90.

9.13.3.11 body_mass_birth

`real(srp) the_body::condition::body_mass_birth`
Body mass at birth, will keep record of it.
Definition at line 55 of file m_body.f90.

9.13.3.12 body_mass_maximum

`real(srp) the_body::condition::body_mass_maximum`
Maximum historically body, will keep record of it.
Definition at line 57 of file m_body.f90.

9.13.3.13 smr

`real(srp) the_body::condition::smr`
Standard metabolitic rate, can change depending on hormones and psychological state (GOS)), at birth initialised from the genome.

Note

SMR is not used in this version.

Definition at line 61 of file m_body.f90.

9.13.3.14 maxstomcap

`real(srp) the_body::condition::maxstomcap = MAX_STOMACH_CAPACITY_DEF`
Maximum stomach capacity, max. fraction of body mass available for food here set from default value. But can change in future versions of the model depending on the body length and the physiological state (so specifically set for each agent rather than defined from global parameter commondata::max_stomach_capacity_def).

Note

In the old model the stomach content cannot surpass maxstomcap=15% of agent's body mass.

Definition at line 69 of file m_body.f90.

9.13.3.15 stomach_content_mass

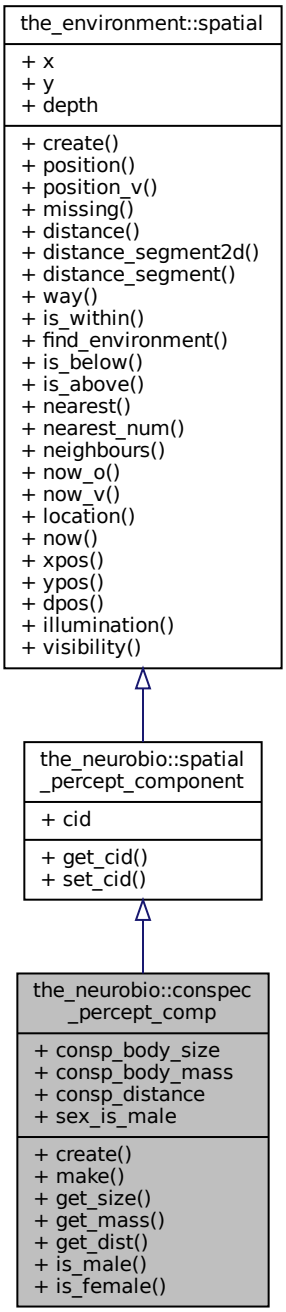
`real(srp) the_body::condition::stomach_content_mass`
Stomach content mass.
Definition at line 71 of file m_body.f90.
The documentation for this type was generated from the following file:

- m_body.f90

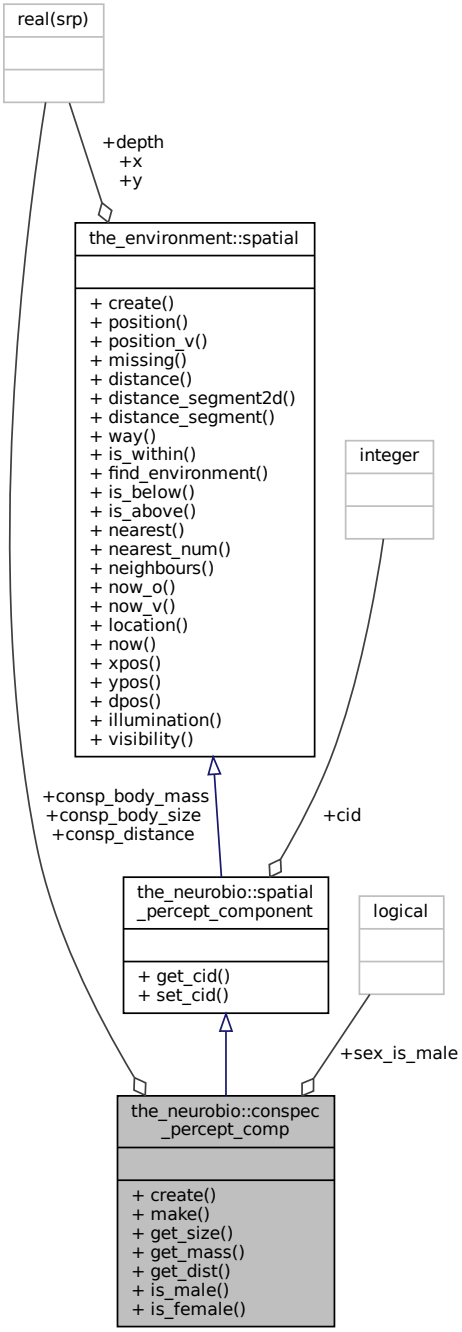
9.14 the_neurobio::conspec_percept_comp Type Reference

This type defines a **single conspecific** perception component. It is required for the `the_neurobio::percept_conspecifics` type that defines the whole conspecifics perception object (array of conspecifics).

Inheritance diagram for the_neurobio::conspec_percept_comp:



Collaboration diagram for the_neurobio::conspec_percept_comp:



Public Member Functions

- procedure, public create => consp_percept_comp_create

  Create a single conspecific perception component at an undefined position with default properties. See `the_neurobio::consp_percept_comp_create()`.

- procedure, public make => consp_percept_make

  Make a single conspecific perception component. This is a single conspecific located within the visual range of the agent. See `the_neurobio::consp_percept_make()`.

- procedure, public get_size => consp_percept_get_size

  *Get the **conspecific** perception component body size. See the_neurobio::consp_percept_get_size().*
- procedure, public get_mass => consp_percept_get_mass

  *Get the **conspecific** perception component body mass. See the_neurobio::consp_percept_get_mass().*
- procedure, public get_dist => consp_percept_get_dist

  *Get the **conspecific** perception component distance. See the_neurobio::consp_percept_get_dist().*
- procedure, public is_male => consp_percept_sex_is_male_get

  *Get the **conspecific** perception component sex flag (male). See the_neurobio::consp_percept_sex_is_male_get().*
- procedure, public is_female => consp_percept_sex_is_female_get

  *Get the **conspecific** perception component sex flag (female). See the_neurobio::consp_percept_sex_is_female_get()*

Public Attributes

- real(srp) consp_body_size

  Body size. The body size of the perception conspecific.
- real(srp) consp_body_mass

  Body mass. The body mass of the perception conspecific.
- real(srp) consp_distance
- logical sex_is_male

  The sex of the conspecific in the perception object.

9.14.1 Detailed Description

This type defines a **single conspecific** perception component. It is required for the the_neurobio::percept_conspecifics type that defines the whole conspecifics perception object (array of conspecifics).
Definition at line 141 of file m_neuro.f90.

9.14.2 Member Function/Subroutine Documentation

9.14.2.1 create()

`procedure, public the_neurobio::conspec_percept_comp::create`
Create a single conspecific perception component at an undefined position with default properties. See the_neurobio::consp_percept_comp_create().
Definition at line 160 of file m_neuro.f90.

9.14.2.2 make()

`procedure, public the_neurobio::conspec_percept_comp::make`
Make a single conspecific perception component. This is a single conspecific located within the visual range of the agent. See the_neurobio::consp_percept_make().
Definition at line 164 of file m_neuro.f90.

9.14.2.3 get_size()

`procedure, public the_neurobio::conspec_percept_comp::get_size`
Get the **conspecific** perception component body size. See the_neurobio::consp_percept_get_size().
Definition at line 167 of file m_neuro.f90.

9.14.2.4 get_mass()

`procedure, public the_neurobio::conspec_percept_comp::get_mass`
Get the **conspecific** perception component body mass. See `the_neurobio::consp_percept_get_mass()`.
Definition at line 170 of file m_neuro.f90.

9.14.2.5 get_dist()

`procedure, public the_neurobio::conspec_percept_comp::get_dist`
Get the **conspecific** perception component distance. See `the_neurobio::consp_percept_get_dist()`.
Definition at line 173 of file m_neuro.f90.

9.14.2.6 is_male()

`procedure, public the_neurobio::conspec_percept_comp::is_male`
Get the **conspecific** perception component sex flag (male). See `the_neurobio::consp_percept_sex_is_male_get()`.
Definition at line 176 of file m_neuro.f90.

9.14.2.7 is_female()

`procedure, public the_neurobio::conspec_percept_comp::is_female`
Get the **conspecific** perception component sex flag (female). See `the_neurobio::consp_percept_sex_is_female_get`
Definition at line 179 of file m_neuro.f90.

9.14.3 Member Data Documentation

9.14.3.1 consp_body_size

`real(srp) the_neurobio::conspec_percept_comp::consp_body_size`
Body size. The body size of the perception conspecific.

Note

We may need body size of the conspecifics as the decision making rules may depend on the conspecific size (e.g. approach big and repulse from small, also may affect mating, e.g. mate with the biggest, etc.).

Definition at line 147 of file m_neuro.f90.

9.14.3.2 consp_body_mass

`real(srp) the_neurobio::conspec_percept_comp::consp_body_mass`
Body mass. The body mass of the perception conspecific.
Definition at line 149 of file m_neuro.f90.

9.14.3.3 consp_distance

`real(srp) the_neurobio::conspec_percept_comp::consp_distance`

Note

Any other characteristics of the perception conspecifics may be added, e.g. sex etc. What is crucial for the decision making. The distance towards this conspecific in the visual field.

Definition at line 153 of file m_neuro.f90.

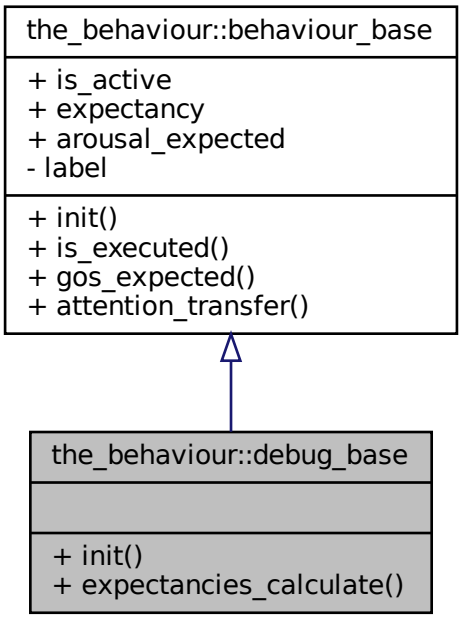
9.14.3.4 sex_is_male

`logical the_neurobio::conspec_percept_comp::sex_is_male`
The sex of the conspecific in the perception object.
Definition at line 155 of file m_neuro.f90.
The documentation for this type was generated from the following file:

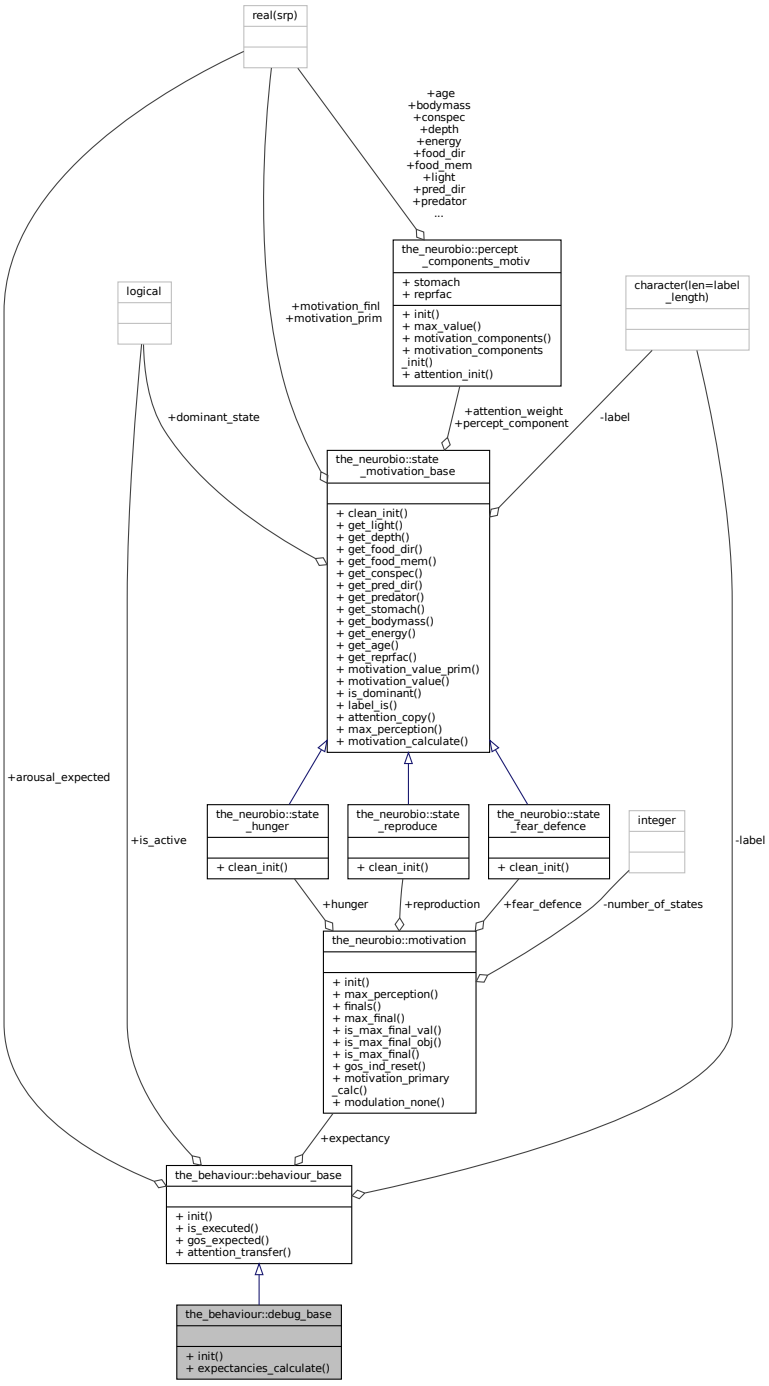
- m_neuro.f90

9.15 the_behaviour::debug_base Type Reference

This is a test fake behaviour unit that is used only for debugging. It cannot be "execute"'d, but the expectancy can be calculated (normally in the debug mode).
Inheritance diagram for the_behaviour::debug_base:



Collaboration diagram for the_behaviour::debug_base:



Public Member Functions

- procedure, public init => debug_base_init_zero
- procedure, public expectancies_calculate => debug_base_motivations_expect

Additional Inherited Members

9.15.1 Detailed Description

This is a test fake behaviour unit that is used only for debugging. It cannot be "execute"'d, but the expectancy can be calculated (normally in the debug mode).
Definition at line 474 of file m_behav.f90.

9.15.2 Member Function/Subroutine Documentation

9.15.2.1 init()

`procedure, public the_behaviour::debug_base::init`
Definition at line 476 of file m_behav.f90.

9.15.2.2 expectancies_calculate()

`procedure, public the_behaviour::debug_base::expectancies_calculate`
Definition at line 477 of file m_behav.f90.
The documentation for this type was generated from the following file:

- m_behav.f90

9.16 the_environment::disassemble Interface Reference

Interface to the procedure to **disassemble** the global habitats objects array the_environment::global_habitats_available back into separate habitat object components. See the_environment::global_habitats_disassemble() for the back-end implementation.
Collaboration diagram for the_environment::disassemble:

Public Member Functions

- subroutine global_habitats_disassemble (habitat_1, habitat_2, habitat_3, habitat_4, habitat_5, habitat_↩ 6, habitat_7, habitat_8, habitat_9, habitat_10, habitat_11, habitat_12, habitat_13, habitat_14, habitat_15, habitat_16, habitat_17, habitat_18, habitat_19, habitat_20, reindex)

  Disassemble the global habitats objects array the_environment::global_habitats_available into separate habitat object.

9.16.1 Detailed Description

Interface to the procedure to **disassemble** the global habitats objects array the_environment::global_habitats_available back into separate habitat object components. See the_environment::global_habitats_disassemble() for the back-end implementation.

Definition at line 806 of file m_env.f90.

9.16.2 Member Function/Subroutine Documentation

9.16.2.1 global_habitats_disassemble()

```
subroutine the_environment::disassemble::global_habitats_disassemble (
             type(habitat), intent(out) habitat_1,
             type(habitat), intent(out), optional habitat_2,
             type(habitat), intent(out), optional habitat_3,
             type(habitat), intent(out), optional habitat_4,
             type(habitat), intent(out), optional habitat_5,
             type(habitat), intent(out), optional habitat_6,
             type(habitat), intent(out), optional habitat_7,
             type(habitat), intent(out), optional habitat_8,
             type(habitat), intent(out), optional habitat_9,
             type(habitat), intent(out), optional habitat_10,
             type(habitat), intent(out), optional habitat_11,
             type(habitat), intent(out), optional habitat_12,
             type(habitat), intent(out), optional habitat_13,
             type(habitat), intent(out), optional habitat_14,
             type(habitat), intent(out), optional habitat_15,
             type(habitat), intent(out), optional habitat_16,
             type(habitat), intent(out), optional habitat_17,
             type(habitat), intent(out), optional habitat_18,
             type(habitat), intent(out), optional habitat_19,
             type(habitat), intent(out), optional habitat_20,
             logical, intent(in), optional reindex )
```

Disassemble the global habitats objects array the_environment::global_habitats_available into separate habitat object.

Parameters

out	*habitat_1*	[inout] habitat_1, ... a list (from 2 to 20) of food resources to restore from the joined state.

Warning

elementary habitats in the list are strictly **type**, extension (class) objects are not supported.

Parameters

in	*reindex*	reindex logical flag to reindex the joined food resource (TRUE) linked to each of the habitats upon disassemble. Default is **no** reindexing.

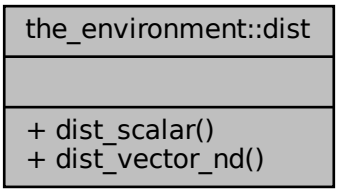
Definition at line 8972 of file m_env.f90.
The documentation for this interface was generated from the following file:

- m_env.f90

9.17 the_environment::dist Interface Reference

Internal distance calculation backend engine.

Collaboration diagram for the_environment::dist:



Public Member Functions

- elemental real(srp) function dist_scalar (x1, x2, y1, y2, z1, z2)

  Calculate distance between 3D or 2D points. This is a function engine for use within type bound procedures. ***Example*** *(dist_scalar):*
- pure real(srp) function dist_vector_nd (cvector1, cvector2)

  Calculate distance between N-dimensional points. This is a function engine for use within other type bound procedures.

9.17.1 Detailed Description

Internal distance calculation backend engine.
Definition at line 757 of file m_env.f90.

9.17.2 Member Function/Subroutine Documentation

9.17.2.1 dist_scalar()

```
elemental real(srp) function the_environment::dist::dist_scalar (
             real(srp), intent(in) x1,
             real(srp), intent(in) x2,
             real(srp), intent(in) y1,
             real(srp), intent(in) y2,
             real(srp), intent(in), optional z1,
             real(srp), intent(in), optional z2 )
```

Calculate distance between 3D or 2D points. This is a function engine for use within type bound procedures.
Example (dist_scalar):

```
dist(1.0,10.0,   2.0,20.0,   3.0,30.0 )
```

Note

This version accepts individual scalar coordinates.

Note that it is an elemental function, accepting also arrays (should have equal size and shape for elemental operations).

Definition at line 5633 of file m_env.f90.

9.17.2.2 dist_vector_nd()

```
pure real(srp) function the_environment::dist::dist_vector_nd (
            real(srp), dimension(:), intent(in) cvector1,
            real(srp), dimension(:), intent(in) cvector2 )
```

Calculate distance between N-dimensional points. This is a function engine for use within other type bound procedures.

Returns

the distance between two N-dimensional vectors.

Parameters

cvector	N-dimensional vectors for the two points we calculate the distance between. For a 3D case the vercors look like: x = cvector(1), y = cvector(2), z = cvector(3). **Example** dist_vector: `dist( [1.0, 2.0, 3.0], [10.0, 20.0, 30.0] )`

Note

This version accepts vectors of coordinates for each of the two objects.

Warning

The shapes and sizes of the two arrays must be equal

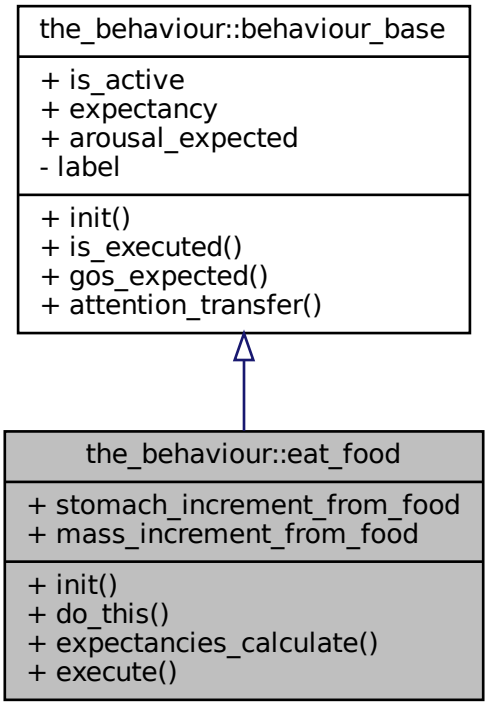
Definition at line 5666 of file m_env.f90.
The documentation for this interface was generated from the following file:

- m_env.f90

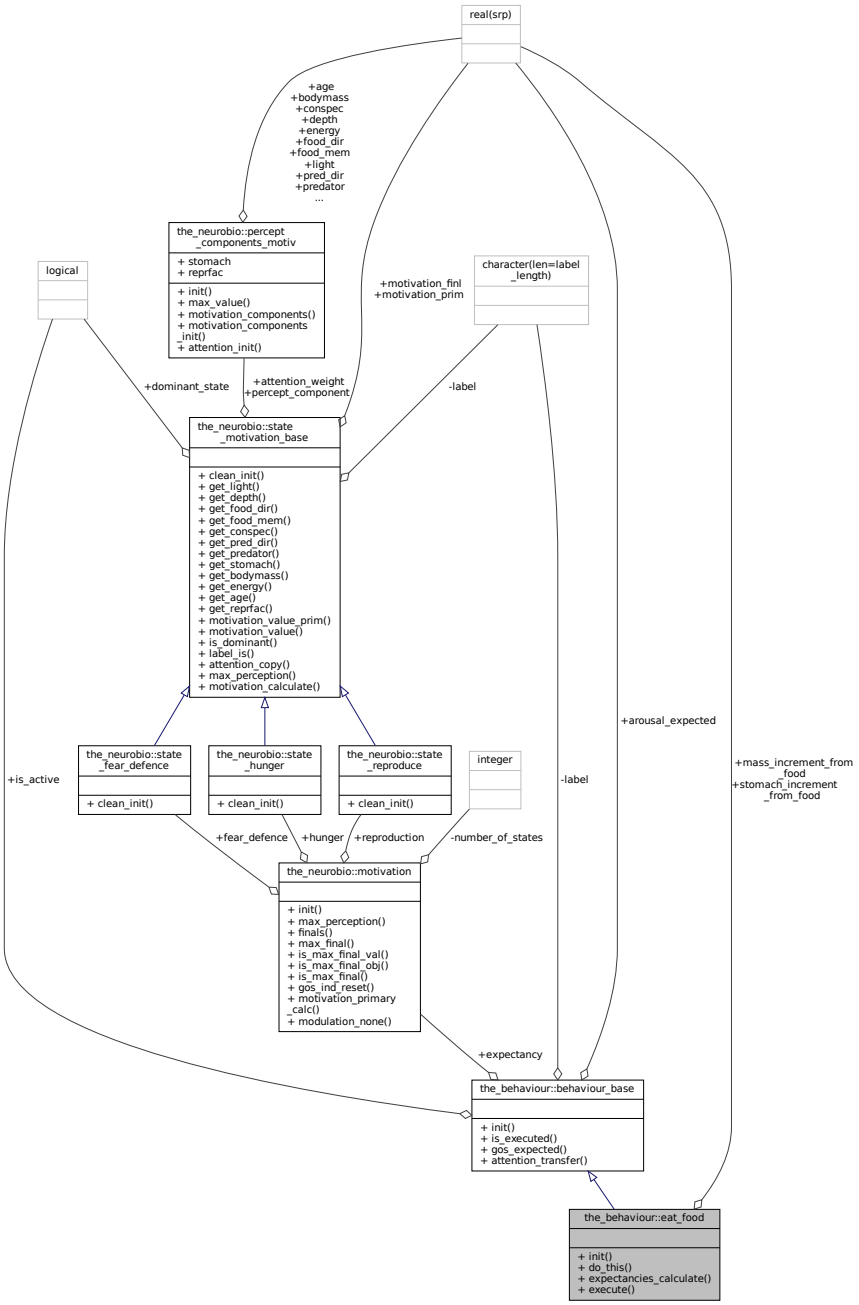
9.18 the_behaviour::eat_food Type Reference

Eat food is consuming food item(s) perceived.

Inheritance diagram for the_behaviour::eat_food:



Collaboration diagram for the_behaviour::eat_food:



Public Member Functions

- procedure, public init => eat_food_item_init_zero

  `init` *inits the behaviour element object. See* `the_behaviour::eat_food_item_init_zero()`.
- procedure, public do_this => eat_food_item_do_this

  `do_this` *performs the agent's action without changing the agent or the environment.*
- procedure, public expectancies_calculate => eat_food_item_motivations_expect

  `expectancies_calculate` *is a subroutine (re)calculating motivations from fake expected perceptions following from* `do_this`.
- procedure, public execute => eat_food_item_do_execute

*execute*performs the action fully, ***changing the state*** *of the agent and the environment. See* *the_behaviour::eat_food_item*

Public Attributes

- real(srp) stomach_increment_from_food

  Increment of the agent's stomach content that results from eating a single specific food item.
- real(srp) mass_increment_from_food

  Increment of the agent's body mass that results from eating a single specific food item.

9.18.1 Detailed Description

Eat food is consuming food item(s) perceived.
Definition at line 112 of file m_behav.f90.

9.18.2 Member Function/Subroutine Documentation

9.18.2.1 init()

`procedure, public the_behaviour::eat_food::init`
`init` inits the behaviour element object. See `the_behaviour::eat_food_item_init_zero()`.
Definition at line 122 of file m_behav.f90.

9.18.2.2 do_this()

`procedure, public the_behaviour::eat_food::do_this`
`do_this` performs the agent's action without changing the agent or the environment.

Warning

`do_this` is **not** intended to be called directly, only from within `expectancies_calculate` and `execute`. See `the_behaviour::eat_food_item_do_this()`.

Definition at line 128 of file m_behav.f90.

9.18.2.3 expectancies_calculate()

`procedure, public the_behaviour::eat_food::expectancies_calculate`
`expectancies_calculate` is a subroutine (re)calculating motivations from fake expected perceptions following from `do_this`.

Note

Note that this is the computational engine to assess the expected GOS of the behaviour, it is called from within the base root behaviour class-bound polymorphic `gos_expect`. See `the_behaviour::eat_food_item_motivatio`

Definition at line 135 of file m_behav.f90.

9.18.2.4 execute()

`procedure, public the_behaviour::eat_food::execute`
`execute`performs the action fully, **changing the state** of the agent and the environment. See `the_behaviour::eat_food_it`
Definition at line 140 of file m_behav.f90.

9.18.3 Member Data Documentation

9.18.3.1 stomach_increment_from_food

`real(srp) the_behaviour::eat_food::stomach_increment_from_food`
Increment of the agent's stomach content that results from eating a single specific food item.
Definition at line 115 of file m_behav.f90.

9.18.3.2 mass_increment_from_food

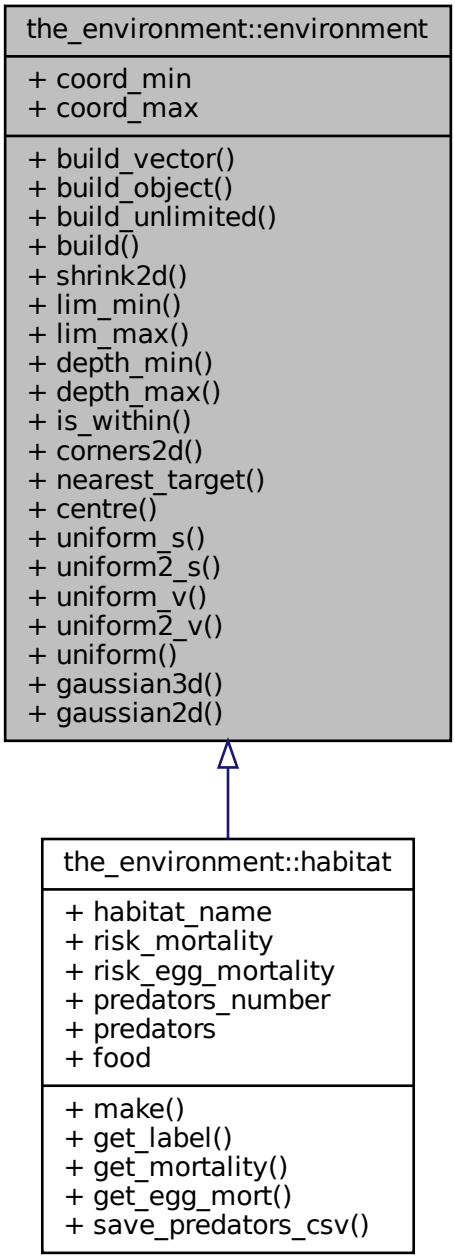
`real(srp) the_behaviour::eat_food::mass_increment_from_food`
Increment of the agent's body mass that results from eating a single specific food item.
Definition at line 118 of file m_behav.f90.
The documentation for this type was generated from the following file:

- m_behav.f90

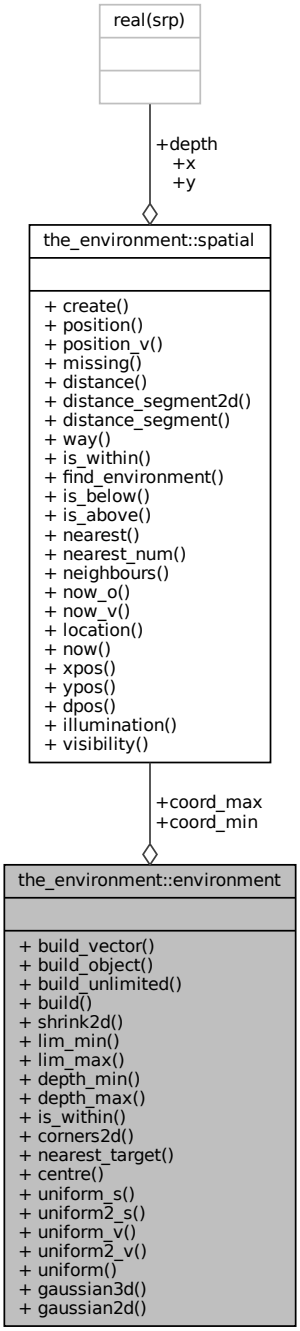
9.19 the_environment::environment Type Reference

Definition of the overall **environment**. Environment is a general container for all habitats, patches and other similar objects where the whole life of the agents takes place. Environment just provides the *limits* for all these objects.

Inheritance diagram for the_environment::environment:



Collaboration diagram for the_environment::environment:



Public Member Functions

- procedure, public build_vector => environment_whole_build_vector

  Create the highest level container environment. Vector-based. See `the_environment::environment_whole_build_vecto`
- procedure, public build_object => environment_whole_build_object

  Create the highest level container environment. Object-based. See `the_environment::environment_whole_build_objec`
- procedure, public build_unlimited => environment_build_unlimited

*Build an **unlimited environment**, with the spatial coordinates limited by the maximum machine supported values based on the intrinsic `huge` function. See `the_environment::environment_build_unlimited()`*

- generic, public build => build_vector, build_object, build_unlimited

  Create the highest level container environment. Generic interface. See `the_environment::environment_whole_build_vec` `the_environment::environment_whole_build_object()` and `the_environment::environment_build_unl`
- procedure, public shrink2d => environment_shrink_xy_fixed

  Return an environment object that is shrunk by a fixed value in the 2D XxY plane. See `the_environment::environment_shrin`
- procedure, public lim_min => environment_get_minimum_obj

  *Function to get the **minimum** spatial limits (coordinates) of the environment. See `the_environment::environment_get_mini`*
- procedure, public lim_max => environment_get_maximum_obj

  *Function to get the **maximum** spatial limits (coordinates) of the environment. See `the_environment::environment_get_maxi`*
- procedure, public depth_min => environment_get_minimum_depth

  *Get the **minimum depth** in this environment. See `the_environment::environment_get_minimum_depth()`.*
- procedure, public depth_max => environment_get_maximum_depth

  *Get the **maximum depth** in this environment. See `the_environment::environment_get_maximum_depth()`.*
- procedure, public is_within => environment_check_located_within_3d

  Check if a spatial object is actually within this environment. See `the_environment::environment_check_located_within`
- procedure, public corners2d => environment_get_corners_2dxy

  Get the corners of the environment in the 2D X Y plane. See `the_environment::environment_get_corners_2dxy()`.
- procedure, public nearest_target => environment_get_nearest_point_in_outside_obj

  Get the spatial point position within this environment that is nearest to an arbitrary spatial object located outside of the this environment. If the spatial object is actually located in this environment,return its own spatial position. See `the_environment::environment_get_nearest_point_in_outside_obj()`.
- procedure, public centre => environment_centre_coordinates_3d

  Determine the centroid of the environment. See `the_environment::environment_centre_coordinates_3d()`
- procedure, public uniform_s => environment_random_uniform_spatial_3d

  *Generate a random spatial object with the uniform distribution within (i.e. bound to) **this** environment. See `the_environment::environment_random_uniform_spatial_3d()`*
- procedure, public uniform2_s => environment_random_uniform_spatial_2d

  *Generate a random spatial object with the uniform distribution within (i.e. bound to) **this** environment, the third depth coordinate is fixed. See `the_environment::environment_random_uniform_spatial_2d()`*
- procedure, public uniform_v =>environment_random_uniform_spatial_vec_3d

  *Generate a vector of random spatial objects with the uniform distribution within (i.e. bound to) **this** environment. Full 3D procedure. See `the_environment::environment_random_uniform_spatial_vec_3d()`*
- procedure, public uniform2_v =>environment_random_uniform_spatial_vec_2d

  *Generate a vector of random spatial objects with the uniform distribution within (i.e. bound to) **this** environment. The third, depth coordinate is non-stochastic, and provided as an array parameter. See `the_environment::environment_random_uniform_spatial_vec_2d()`*
- generic, public uniform => uniform_s, uniform2_s, uniform_v, uniform2_v

  *Generate a vector of random spatial objects with the uniform distribution within (i.e. bound to) **this** environment. Generic interface.*
- procedure, public gaussian3d => environment_random_gaussian_spatial_3d

  *Generates a vector of random spatial object with Gaussian coordinates within (i.e. bound to) **this** environment. Full 3D procedure. See `the_environment::environment_random_gaussian_spatial_3d()`*
- procedure, public gaussian2d => environment_random_gaussian_spatial_2d

  *Generates a vector of random spatial object with Gaussian coordinates within (i.e. bound to) **this** environment. The depth coordinate is set separately and can be non-random (fixed for the whole output array) or Gaussian with separate variance. See `the_environment::environment_random_gaussian_spatial_2d()`*

Public Attributes

- type(spatial) coord_min

  Set shape and limits of the whole environment, by default a rectangle with Cartesian coordinates based on ENVIRONMENT_WHOLE_SIZE. The minimum and maximum coordinates are set through the `SPATIAL` object.
- type(spatial) coord_max

9.19.1 Detailed Description

Definition of the overall **environment**. Environment is a general container for all habitats, patches and other similar objects where the whole life of the agents takes place. Environment just provides the *limits* for all these objects.

Warning

In this version, the environment objects are the most simplistic form: 3D "boxes". An arbitrary convex *polyhedron*-based environment can be implemented but this requires a more complex computational geometry backend.

Note

Coordinate system should **always** use the `SPATIAL` type objects, we don't define _v-type procedures (it is also unclear are _v procedures really necessary).

Definition at line 277 of file m_env.f90.

9.19.2 Member Function/Subroutine Documentation

9.19.2.1 build_vector()

`procedure, public the_environment::environment::build_vector`
Create the highest level container environment. Vector-based. See `the_environment::environment_whole_build_vec`
Definition at line 286 of file m_env.f90.

9.19.2.2 build_object()

`procedure, public the_environment::environment::build_object`
Create the highest level container environment. Object-based. See `the_environment::environment_whole_build_obj`
Definition at line 289 of file m_env.f90.

9.19.2.3 build_unlimited()

`procedure, public the_environment::environment::build_unlimited`
Build an **unlimited environment**, with the spatial coordinates limited by the maximum machine supported values based on the intrinsic `huge` function. See `the_environment::environment_build_unlimited()`
Definition at line 294 of file m_env.f90.

9.19.2.4 build()

`generic, public the_environment::environment::build`
Create the highest level container environment. Generic interface. See `the_environment::environment_whole_build_v` `the_environment::environment_whole_build_object()` and `the_environment::environment_build_u`
Definition at line 299 of file m_env.f90.

9.19.2.5 shrink2d()

`procedure, public the_environment::environment::shrink2d`
Return an environment object that is shrunk by a fixed value in the 2D XxY plane. See `the_environment::environment_shr`
Definition at line 303 of file m_env.f90.

9.19.2.6 lim_min()

procedure, public the_environment::environment::lim_min
Function to get the **minimum** spatial limits (coordinates) of the environment. See the_environment::environment_get_mi
Definition at line 307 of file m_env.f90.

9.19.2.7 lim_max()

procedure, public the_environment::environment::lim_max
Function to get the **maximum** spatial limits (coordinates) of the environment. See the_environment::environment_get_ma
Definition at line 311 of file m_env.f90.

9.19.2.8 depth_min()

procedure, public the_environment::environment::depth_min
Get the **minimum depth** in this environment. See the_environment::environment_get_minimum_depth().
Definition at line 314 of file m_env.f90.

9.19.2.9 depth_max()

procedure, public the_environment::environment::depth_max
Get the **maximum depth** in this environment. See the_environment::environment_get_maximum_depth().
Definition at line 317 of file m_env.f90.

9.19.2.10 is_within()

procedure, public the_environment::environment::is_within
Check if a spatial object is actually within this environment. See the_environment::environment_check_located_wit
Definition at line 320 of file m_env.f90.

9.19.2.11 corners2d()

procedure, public the_environment::environment::corners2d
Get the corners of the environment in the 2D X Y plane. See the_environment::environment_get_corners_2dxy().
Definition at line 323 of file m_env.f90.

9.19.2.12 nearest_target()

procedure, public the_environment::environment::nearest_target
Get the spatial point position within this environment that is nearest to an arbitrary spatial object located outside of the this environment. If the spatial object is actually located in this environment,return its own spatial position. See the_environment::environment_get_nearest_point_in_outside_obj().
Definition at line 329 of file m_env.f90.

9.19.2.13 centre()

procedure, public the_environment::environment::centre
Determine the centroid of the environment. See the_environment::environment_centre_coordinates_3d()
Definition at line 333 of file m_env.f90.

9.19.2.14 uniform_s()

procedure, public the_environment::environment::uniform_s
Generate a random spatial object with the uniform distribution within (i.e. bound to) **this** environment. See the_environment::environment_random_uniform_spatial_3d()
Definition at line 337 of file m_env.f90.

9.19.2.15 uniform2_s()

procedure, public the_environment::environment::uniform2_s
Generate a random spatial object with the uniform distribution within (i.e. bound to) **this** environment, the third depth coordinate is fixed. See the_environment::environment_random_uniform_spatial_2d()
Definition at line 342 of file m_env.f90.

9.19.2.16 uniform_v()

procedure, public the_environment::environment::uniform_v
Generate a vector of random spatial objects with the uniform distribution within (i.e. bound to) **this** environment. Full 3D procedure. See the_environment::environment_random_uniform_spatial_vec_3d()
Definition at line 346 of file m_env.f90.

9.19.2.17 uniform2_v()

procedure, public the_environment::environment::uniform2_v
Generate a vector of random spatial objects with the uniform distribution within (i.e. bound to) **this** environment. The third, depth coordinate is non-stochastic, and provided as an array parameter. See the_environment::environment_random_uniform_spatial_vec_2d()
Definition at line 351 of file m_env.f90.

9.19.2.18 uniform()

generic, public the_environment::environment::uniform
Generate a vector of random spatial objects with the uniform distribution within (i.e. bound to) **this** environment. Generic interface.
Definition at line 355 of file m_env.f90.

9.19.2.19 gaussian3d()

procedure, public the_environment::environment::gaussian3d
Generates a vector of random spatial object with Gaussian coordinates within (i.e. bound to) **this** environment. Full 3D procedure. See the_environment::environment_random_gaussian_spatial_3d()
Definition at line 359 of file m_env.f90.

9.19.2.20 gaussian2d()

procedure, public the_environment::environment::gaussian2d
Generates a vector of random spatial object with Gaussian coordinates within (i.e. bound to) **this** environment. The depth coordinate is set separately and can be non-random (fixed for the whole output array) or Gaussian with separate variance. See the_environment::environment_random_gaussian_spatial_2d()
Definition at line 365 of file m_env.f90.

9.19.3 Member Data Documentation

9.19.3.1 coord_min

`type(spatial) the_environment::environment::coord_min`
Set shape and limits of the whole environment, by default a rectangle with Cartesian coordinates based on ENVIRONMENT_WHOLE_SIZE. The minimum and maximum coordinates are set through the `SPATIAL` object.
Definition at line 282 of file m_env.f90.

9.19.3.2 coord_max

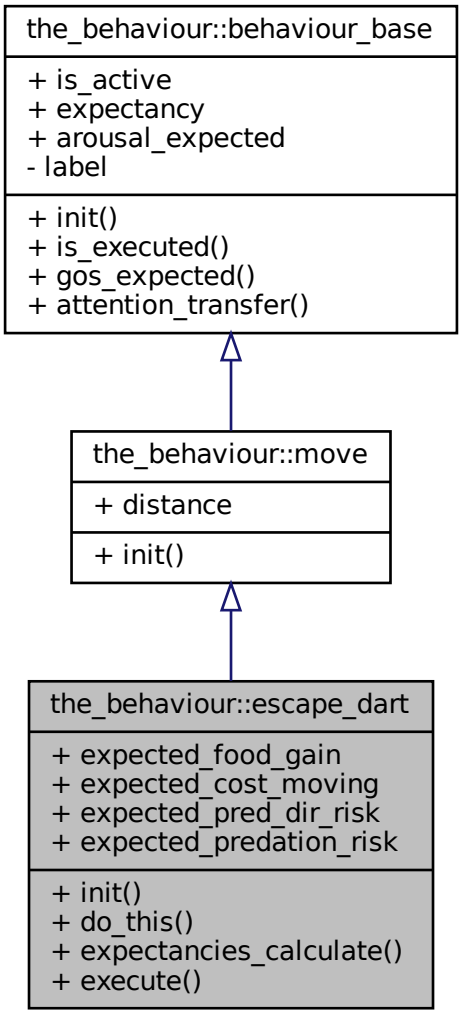
`type(spatial) the_environment::environment::coord_max`
Definition at line 282 of file m_env.f90.
The documentation for this type was generated from the following file:

- m_env.f90

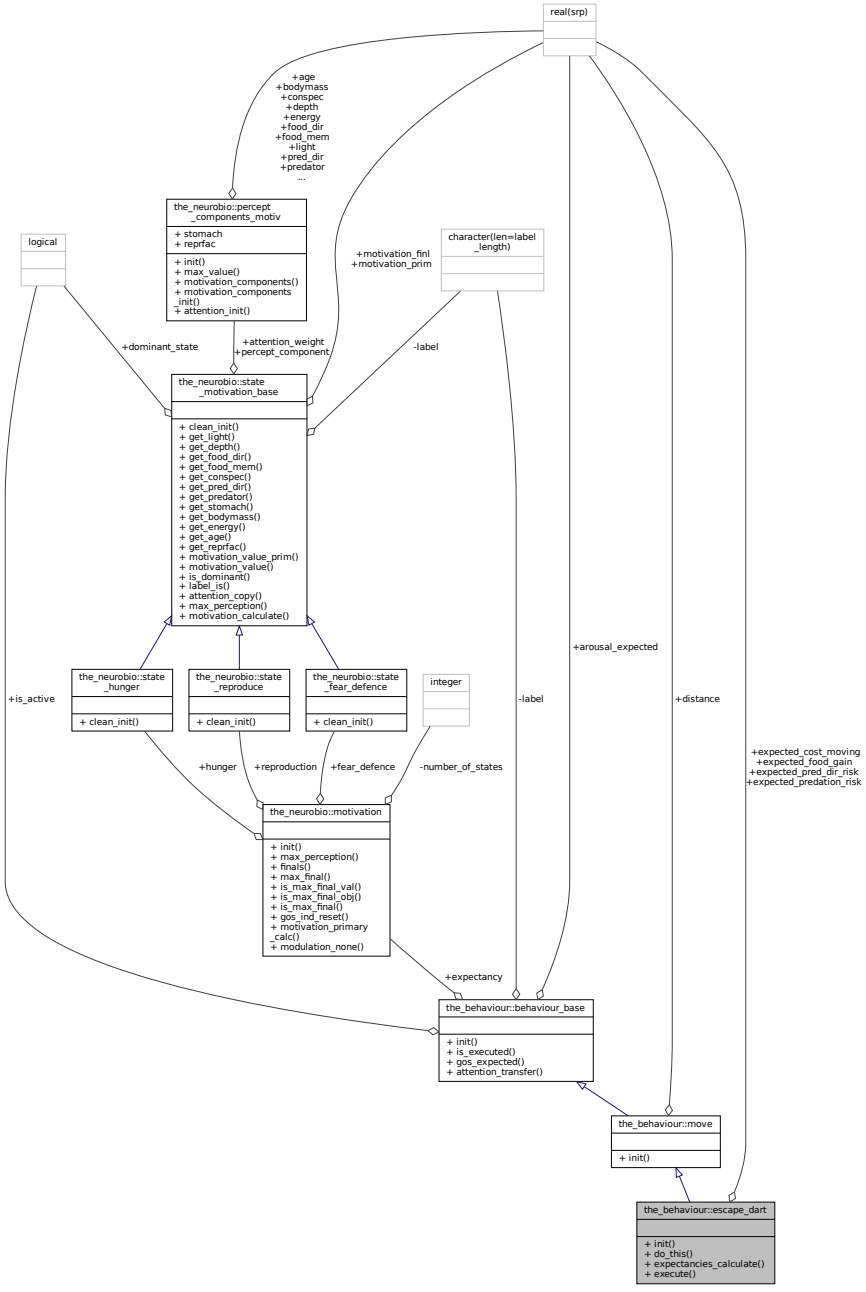
9.20 the_behaviour::escape_dart Type Reference

Escape dart is a very fast long distance movement, normally in response to a direct predation threat.

Inheritance diagram for the_behaviour::escape_dart:



Collaboration diagram for the_behaviour::escape_dart:



Public Member Functions

- procedure, public init => escape_dart_init_zero

  *Initialise the **escape dart** behaviour component to a zero state. Dart is a quick high speed active escape. See* `the_behaviour::escape_dart_init_zero()`.

- procedure, public do_this => escape_dart_do_this

  Do active escape dart by `this_agent` *(the actor agent). Subjective assessment of the motivational value for this is based on the distance of escape (in turn, dependent on the visibility of the predator). See* `the_behaviour::escape_dart_do_this()`.

- procedure, public expectancies_calculate => escape_dart_motivations_expect

  `escape_dart::motivations_expect()` *is a subroutine (re)calculating motivations from fake expected perceptions following from* `escape_dart::do_this()` => `the_behaviour::escape_dart_do_this()`.

See `the_behaviourescape_dart_motivations_expect()

- procedure, public execute => escape_dart_do_execute

  Execute this behaviour component "escape" by `this_agent` agent. See `the_behaviour::escape_dart_do_execute()`.

Public Attributes

- real(srp) expected_food_gain

  *The expected food gain (body mass increment) is always **null** for active escape.*

- real(srp) expected_cost_moving

  Expected body mass cost of movement; depends on the distance. Distance, in turn, should be calculated based on the visual range detectability of the predator for the agent, in the do_this procedure.

- real(srp) expected_pred_dir_risk

  The expected direct predation risk is zero for active escape.

- real(srp) expected_predation_risk

  The expected general predation risk, i.e. the risk depending on the current number of predators in both the perception and memory stack. The expected risk assumes that a escape moves the agent fully out of reach of any direct predation risk.

9.20.1 Detailed Description

Escape dart is a very fast long distance movement, normally in response to a direct predation threat.
Definition at line 251 of file m_behav.f90.

9.20.2 Member Function/Subroutine Documentation

9.20.2.1 init()

`procedure, public the_behaviour::escape_dart::init`
Initialise the **escape dart** behaviour component to a zero state. Dart is a quick high speed active escape. See `the_behaviour::escape_dart_init_zero()`.
Definition at line 270 of file m_behav.f90.

9.20.2.2 do_this()

`procedure, public the_behaviour::escape_dart::do_this`
Do active escape dart by `this_agent` (the actor agent). Subjective assessment of the motivational value for this is based on the distance of escape (in turn, dependent on the visibility of the predator). See `the_behaviour::escape_dart_do_this()`.
Definition at line 276 of file m_behav.f90.

9.20.2.3 expectancies_calculate()

`procedure, public the_behaviour::escape_dart::expectancies_calculate`
`escape_dart::motivations_expect()` is a subroutine (re)calculating motivations from fake expected perceptions following from `escape_dart::do_this()` => `the_behaviour::escape_dart_do_this()`.
See `the_behaviourescape_dart_motivations_expect()
Definition at line 281 of file m_behav.f90.

9.20.2.4 execute()

`procedure, public the_behaviour::escape_dart::execute`
Execute this behaviour component "escape" by `this_agent` agent. See `the_behaviour::escape_dart_do_execute()`
Definition at line 285 of file m_behav.f90.

9.20.3 Member Data Documentation

9.20.3.1 expected_food_gain

`real(srp) the_behaviour::escape_dart::expected_food_gain`
The expected food gain (body mass increment) is always **null** for active escape.
Definition at line 254 of file m_behav.f90.

9.20.3.2 expected_cost_moving

`real(srp) the_behaviour::escape_dart::expected_cost_moving`
Expected body mass cost of movement; depends on the distance. Distance, in turn, should be calculated based on the visual range detectability of the predator for the agent, in the do_this procedure.
Definition at line 258 of file m_behav.f90.

9.20.3.3 expected_pred_dir_risk

`real(srp) the_behaviour::escape_dart::expected_pred_dir_risk`
The expected direct predation risk is zero for active escape.
Definition at line 260 of file m_behav.f90.

9.20.3.4 expected_predation_risk

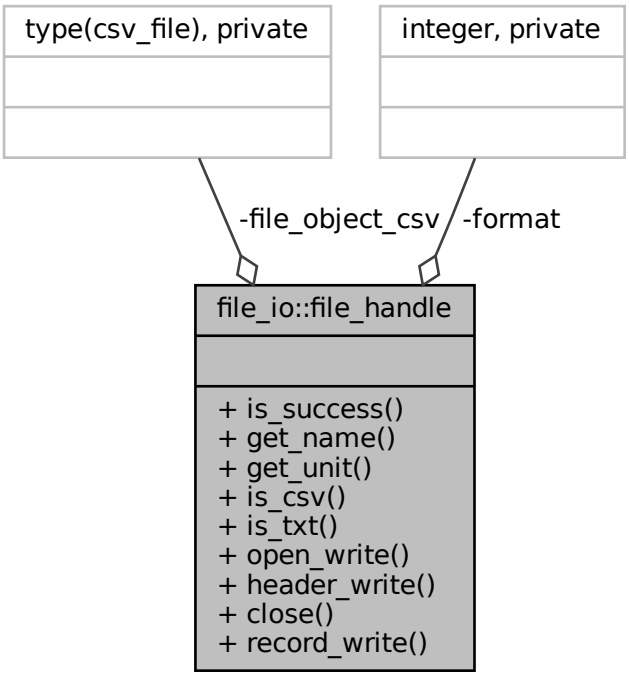
`real(srp) the_behaviour::escape_dart::expected_predation_risk`
The expected general predation risk, i.e. the risk depending on the current number of predators in both the perception and memory stack. The expected risk assumes that a escape moves the agent fully out of reach of any direct predation risk.
Definition at line 265 of file m_behav.f90.
The documentation for this type was generated from the following file:

- m_behav.f90

9.21 file_io::file_handle Type Reference

`FILE_HANDLE` is the basic file handle object. It provides an unitary object oriented interface for operations with any supported file types.

Collaboration diagram for file_io::file_handle:



Public Member Functions

- procedure, public is_success => file_operation_last_is_success

  Obtain the success (TRUE) or failure (FALSE) status of the latest file operation. See example code.
- procedure, public get_name => file_hangle_get_name_string

  Get the file name associated with the file handle. See example code.
- procedure, public get_unit => file_object_get_associated_unit

  Get the Fortran unit number associated with the file handle object. See example code.
- procedure, public is_csv => file_object_format_is_csv

  Check if the file format is CSV. See example code.
- procedure, public is_txt => file_object_format_is_txt

  Check if the file format is TXT. See example code.
- procedure, public open_write => csv_open_write_this

  Open file for physical writing on the disk. See example code.
- procedure, public header_write => csv_header_line_write_this

  Physically write an information header (metadata) about the file. See example code.
- procedure, public close => csv_close_this

  Closes the file for reading or writing. See example code.
- procedure, public record_write => csv_record_string_write_this

  Physically write a single string CSV data record (row) to the file. See example code.

Private Attributes

- integer, private format

  The format of the file, i.e. the file type. The following file types are implemented:

- type(csv_file), private file_object_csv

  file_object_csv is the standard CSV_FILE type from CSV_IO module HEDTOOLS. For details on this component type see CSV_FILE CSV_FILE has the following sub-components:

9.21.1 Detailed Description

FILE_HANDLE is the basic file handle object. It provides an unitary object oriented interface for operations with any supported file types.

Example:

```
type(FILE_HANDLE) :: data_file
```

Only CSV format is supported so far.

Note

It would be even more useful to add the CSV file record into the type as an allocatable component and implement simple record_clean record_append functions. However, allocatable components are not supported in GNU gfortran 4.8, issues error: 'Error: Deferred-length character component 'record_↩ string' at (1) is not yet supported`. Therefore, raw operations with the CSV file record are left here as in the non-OO versions. See extended example in HEDTOOLS and the code of this procedure: the_population::population_save_data_all_agents_csv(). But note that these codes use standard non-object-oriented procedures from HEDTOOLS, not these object oriented wrappers.

Definition at line 148 of file m_fileio.f90.

9.21.2 Member Function/Subroutine Documentation

9.21.2.1 is_success()

```
procedure, public file_io::file_handle::is_success
```

Obtain the success (TRUE) or failure (FALSE) status of the latest file operation. See example code.

Example:

```
if ( data_file%is_success() ) then
```

Definition at line 177 of file m_fileio.f90.

9.21.2.2 get_name()

```
procedure, public file_io::file_handle::get_name
```

Get the file name associated with the file handle. See example code.

Example:

```
print *, data_file%get_name()
```

Definition at line 185 of file m_fileio.f90.

9.21.2.3 get_unit()

```
procedure, public file_io::file_handle::get_unit
```

Get the Fortran unit number associated with the file handle object. See example code.

Example:

```
print *, data_file%get_unit()
```

Definition at line 194 of file m_fileio.f90.

9.21.2.4 is_csv()

`procedure, public file_io::file_handle::is_csv`
Check if the file format is CSV. See example code.
Definition at line 197 of file m_fileio.f90.

9.21.2.5 is_txt()

`procedure, public file_io::file_handle::is_txt`
Check if the file format is TXT. See example code.
Definition at line 200 of file m_fileio.f90.

9.21.2.6 open_write()

`procedure, public file_io::file_handle::open_write`
Open file for physical writing on the disk. See example code.
Example:
```
call data_file%open_write( "file_001.csv" )
```
Definition at line 208 of file m_fileio.f90.

9.21.2.7 header_write()

`procedure, public file_io::file_handle::header_write`
Physically write an information header (metadata) about the file. See example code.
Example:
```
call data_file%header_write("Agent data at start of the simulation")
```
Definition at line 216 of file m_fileio.f90.

9.21.2.8 close()

`procedure, public file_io::file_handle::close`
Closes the file for reading or writing. See example code.
Example:
```
call data_file%close()
```
Definition at line 224 of file m_fileio.f90.

9.21.2.9 record_write()

`procedure, public file_io::file_handle::record_write`
Physically write a single string CSV data record (row) to the file. See example code.
Example:
```
call data_file%record_write( record_string )
```
Definition at line 232 of file m_fileio.f90.

9.21.3 Member Data Documentation

9.21.3.1 format

`integer, private file_io::file_handle::format  [private]`
The format of the file, i.e. the file type. The following file types are implemented:

- `FORMAT_CSV` – Comma Separated Values is a plain text;
- `FORMAT_TXT` – Plain text file.

Definition at line 154 of file m_fileio.f90.

9.21.3.2 file_object_csv

`type (csv_file), private file_io::file_handle::file_object_csv [private]`
`file_object_csv` is the standard `CSV_FILE` type from CSV_IO module HEDTOOLS. For details on this component type see `CSV_FILE` `CSV_FILE` has the following sub-components:

- `character (len=MAX_FILENAME) :: name` – file name;
- `integer :: unit` – Fortran file unit;
- `logical :: status` – Logical status of the last operation.

Note

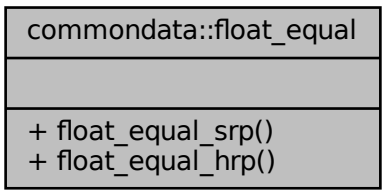
In the wrapper implementation routines, just substitute the original CSV file handle object with thisfile_object↩_csv.

Definition at line 165 of file m_fileio.f90.
The documentation for this type was generated from the following file:

- m_fileio.f90

9.22 commondata::float_equal Interface Reference

Check if two real values are nearly equal using the commondata::is_near_zero(). Thus function can be used for comparing two real values like this:
Collaboration diagram for commondata::float_equal:



Public Member Functions

- elemental logical function float_equal_srp (value1, value2, epsilon)

  Check if two real values are nearly equal using the commondata::is_near_zero(). Thus function can be used for comparing two real values like the below. The exact comparison (incorrect due to possible rounding):
- elemental logical function float_equal_hrp (value1, value2, epsilon)

  Check if two real values are nearly equal using the commondata::is_near_zero(). Thus function can be used for comparing two real values like the below. The exact comparison (incorrect due to possible rounding):

9.22.1 Detailed Description

Check if two real values are nearly equal using the commondata::is_near_zero(). Thus function can be used for comparing two real values like this:

- The exact float comparison (incorrect due to possible rounding):
  `if ( a == b ) ...`
- should be substituted by such comparison:
  `if ( float_equal(a, b) ) ...`

See the backend procedures commondata::float_equal_srp() and commondata::float_equal_hrp() for details.
Definition at line 5406 of file m_common.f90.

9.22.2 Member Function/Subroutine Documentation

9.22.2.1 float_equal_srp()

```
elemental logical function commondata::float_equal::float_equal_srp (
            real(srp), intent(in) value1,
            real(srp), intent(in) value2,
            real(srp), intent(in), optional epsilon )
```

Check if two real values are nearly equal using the commondata::is_near_zero(). Thus function can be used for comparing two real values like the below. The exact comparison (incorrect due to possible rounding):
`if ( a == b ) ...`
should be substituted by such comparison:
`if ( float_equal(a, b, 0.00001) ) ...`
There is also a user defined operator `.feq.` for approximate float point equality. It differs from this function in that the later allows to set an arbitrary epsilon tolerance value whereas the operator does not (it uses the default epsilon based on the intrinsic `tiny()` function.

Note

This is the **standard** precision function (commondata::srp).

Parameters

in	*value1*	value1 The first value for comparison.
in	*value2*	value2 The second value for comparison.
in	*epsilon*	epsilon optional (very small) tolerance value.

Returns

TRUE if the values `value1` and `value2` are nearly equal.

Definition at line 6268 of file m_common.f90.

9.22.2.2 float_equal_hrp()

```
elemental logical function commondata::float_equal::float_equal_hrp (
            real(hrp), intent(in) value1,
            real(hrp), intent(in) value2,
            real(hrp), intent(in), optional epsilon )
```

Check if two real values are nearly equal using the commondata::is_near_zero(). Thus function can be used for comparing two real values like the below. The exact comparison (incorrect due to possible rounding):
`if ( a == b ) ...`
should be substituted by such comparison:
`if ( float_equal(a, b, 0.00001) ) ...`
There is also a user defined operator `.feq.` for approximate float point equality. It differs from this function in that the later allows to set an arbitrary epsilon tolerance value whereas the operator does not (it uses the default epsilon based on the intrinsic `tiny()` function.

Note

This is the **high** precision function (commondata::hrp).

Parameters

in	*value1*	value1 The first value for comparison.
in	*value2*	value2 The second value for comparison.
in	*epsilon*	epsilon optional (very small) tolerance value.

Returns

TRUE if the values `value1` and `value2` are nearly equal.

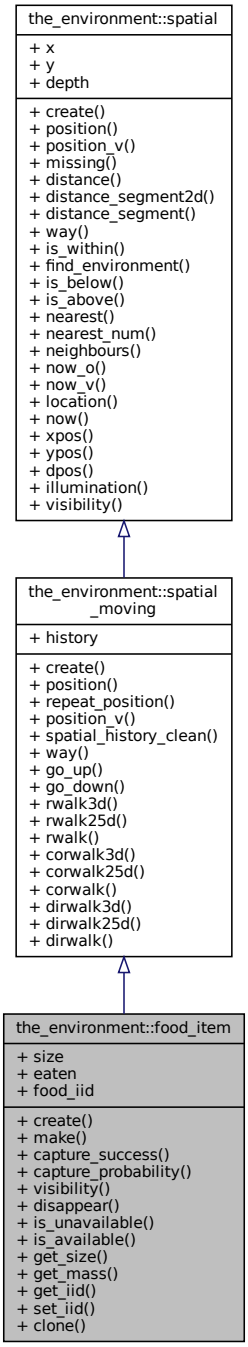
Definition at line 6306 of file m_common.f90.
The documentation for this interface was generated from the following file:

- m_common.f90

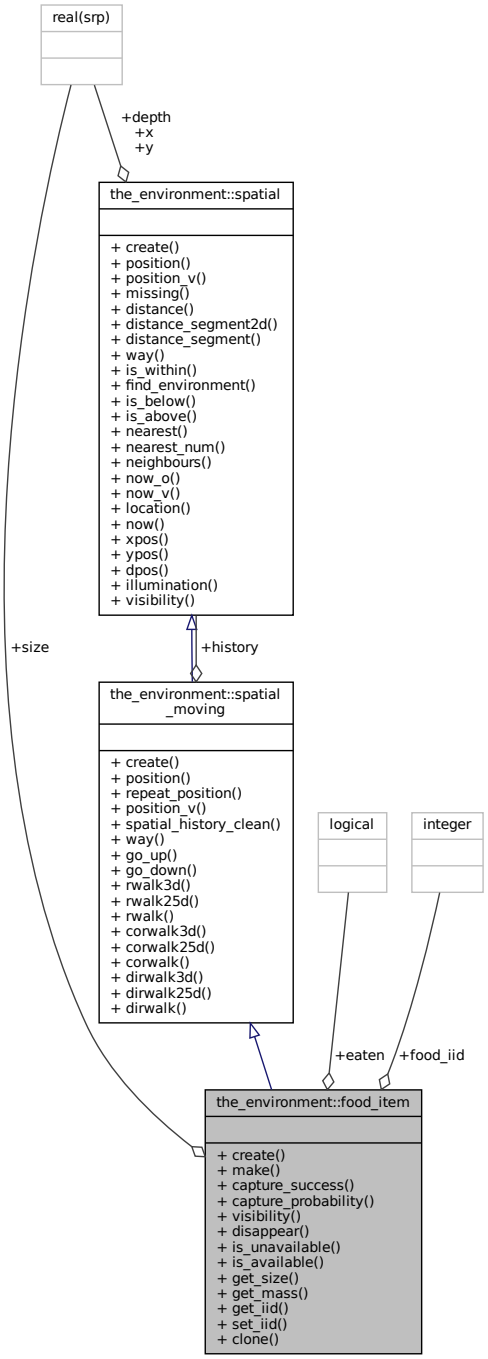
9.23 the_environment::food_item Type Reference

Definition of a single food item. Food item is a spatial object that has specific location in space. It can be "created" and eaten ("disappear"). Food item is an immobile SPATIAL object that has a position in 3D space.

Inheritance diagram for the_environment::food_item:



Collaboration diagram for the_environment::food_item:



Public Member Functions

- procedure, public create => food_item_create

  Create a single food item at an undefined position with default size. See `the_environment::food_item_create()`
- procedure, public make => food_item_make

  Make a single food item, i.e. place it into a specific position in the model environment space and set the size. See `the_environment::food_item_make()`
- procedure, public capture_success => food_item_capture_success_stochast

*Stochastic outcome of **this** food item capture by an agent. Returns TRUE if the food item is captured. See `the_environment::food_item_capture_success_stochast()`*

- procedure, public capture_probability => food_item_capture_probability_calc

  *Calculate the probability of capture of **this** food item by a predator agent depending on the distance between the agent and this food item. See `the_environment::food_item_capture_probability_calc()`*

- procedure, public visibility => food_item_visibility_visual_range

  Calculate the visibility range of this food item. Wrapper to the `visual_range` function. This function calculates the distance from which this food item can be seen by a predator (i.e. the default predator's visual range). See `the_environment::food_item_visibility_visual_range()`

- procedure, public disappear => food_item_disappear

  Make the food item "disappear" and take the "eaten" state, i.e. impossible for consumption by the agents. See `the_environment::food_item_disappear()`

- procedure, public is_unavailable => food_item_is_eaten_unavailable

  Logical check-indicator function for the food item being eaten and not available. See `the_environment::food_item_is_eaten_`

- procedure, public is_available => food_item_is_available

  Logical check-indicator function for the food item being available.

- procedure, public get_size => food_item_get_size

  Get the size component of the food item object. See `the_environment::food_item_get_size()`

- procedure, public get_mass => food_item_get_mass

  Calculate and get the mass of the food item. See `the_environment::food_item_get_mass()`

- procedure, public get_iid => food_item_get_iid

  Get the unique id of the food item object. See `the_environment::food_item_get_iid()`

- procedure, public set_iid => food_item_set_iid

  Set unique id for the food item object. See `the_environment::food_item_set_iid()`

- procedure, public clone => food_item_clone_assign

  Clone the properties of this food item to another food item. See `the_environment::food_item_clone_assign()`

Public Attributes

- real(srp) size

  Food item has a size (radius) that determines its visibility and nutritional value for the predatory agent.

- logical eaten

  Food item can be present or absent (eaten by the agent, =.TRUE.).

- integer food_iid

  Unique ID of this food item. Needed in the resource array.

9.23.1 Detailed Description

Definition of a single food item. Food item is a spatial object that has specific location in space. It can be "created" and eaten ("disappear"). Food item is an immobile SPATIAL object that has a position in 3D space.
Definition at line 371 of file m_env.f90.

9.23.2 Member Function/Subroutine Documentation

9.23.2.1 create()

`procedure, public the_environment::food_item::create`
Create a single food item at an undefined position with default size. See `the_environment::food_item_create()`
Definition at line 382 of file m_env.f90.

9.23.2.2 make()

procedure, public the_environment::food_item::make
Make a single food item, i.e. place it into a specific position in the model environment space and set the size. See the_environment::food_item_make()
Definition at line 386 of file m_env.f90.

9.23.2.3 capture_success()

procedure, public the_environment::food_item::capture_success
Stochastic outcome of **this** food item capture by an agent. Returns TRUE if the food item is captured. See the_environment::food_item_capture_success_stochast()
Definition at line 390 of file m_env.f90.

9.23.2.4 capture_probability()

procedure, public the_environment::food_item::capture_probability
Calculate the probability of capture of **this** food item by a predator agent depending on the distance between the agent and this food item. See the_environment::food_item_capture_probability_calc()
Definition at line 394 of file m_env.f90.

9.23.2.5 visibility()

procedure, public the_environment::food_item::visibility
Calculate the visibility range of this food item. Wrapper to the visual_range function. This function calculates the distance from which this food item can be seen by a predator (i.e. the default predator's visual range). See the_environment::food_item_visibility_visual_range()
Definition at line 400 of file m_env.f90.

9.23.2.6 disappear()

procedure, public the_environment::food_item::disappear
Make the food item "disappear" and take the "eaten" state, i.e. impossible for consumption by the agents. See the_environment::food_item_disappear()
Definition at line 404 of file m_env.f90.

9.23.2.7 is_unavailable()

procedure, public the_environment::food_item::is_unavailable
Logical check-indicator function for the food item being eaten and not available. See the_environment::food_item_is_eat
Definition at line 408 of file m_env.f90.

9.23.2.8 is_available()

procedure, public the_environment::food_item::is_available
Logical check-indicator function for the food item being available.

Returns

Logical indicator TRUE if the food item is present in the environment and therefore available. See the_environment::food_item_is_available()

Definition at line 413 of file m_env.f90.

9.23.2.9 get_size()

procedure, public the_environment::food_item::get_size
Get the size component of the food item object. See the_environment::food_item_get_size()
Definition at line 416 of file m_env.f90.

9.23.2.10 get_mass()

procedure, public the_environment::food_item::get_mass
Calculate and get the mass of the food item. See the_environment::food_item_get_mass()
Definition at line 419 of file m_env.f90.

9.23.2.11 get_iid()

procedure, public the_environment::food_item::get_iid
Get the unique id of the food item object. See the_environment::food_item_get_iid()
Definition at line 422 of file m_env.f90.

9.23.2.12 set_iid()

procedure, public the_environment::food_item::set_iid
Set unique id for the food item object. See the_environment::food_item_set_iid()
Definition at line 425 of file m_env.f90.

9.23.2.13 clone()

procedure, public the_environment::food_item::clone
Clone the properties of this food item to another food item. See the_environment::food_item_clone_assign()
Definition at line 428 of file m_env.f90.

9.23.3 Member Data Documentation

9.23.3.1 size

real(srp) the_environment::food_item::size
Food item has a size (radius) that determines its visibility and nutritional value for the predatory agent.
Definition at line 374 of file m_env.f90.

9.23.3.2 eaten

logical the_environment::food_item::eaten
Food item can be present or absent (eaten by the agent, =.TRUE.).
Definition at line 376 of file m_env.f90.

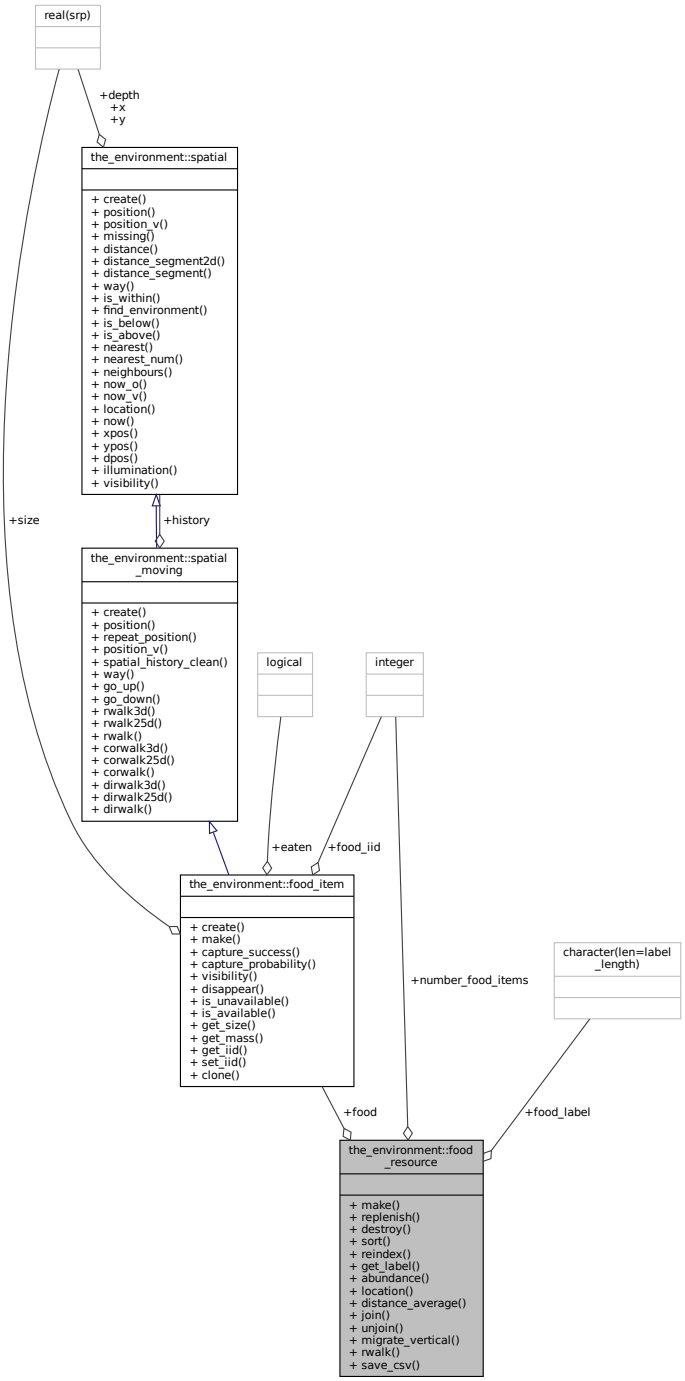
9.23.3.3 food_iid

integer the_environment::food_item::food_iid
Unique ID of this food item. Needed in the resource array.
Definition at line 378 of file m_env.f90.
The documentation for this type was generated from the following file:

- m_env.f90

9.24 the_environment::food_resource Type Reference

Definition of the super-type FOOD resource type. This is a superclass, several sub-classes can be defined for different kinds of food and prey objects.
Collaboration diagram for the_environment::food_resource:



Public Member Functions

- procedure, public make => food_resource_make

  Make food resource object. This class standard constructor. See `the_environment::food_resource_make()`

- procedure, public replenish => food_resource_replenish_food_items_all

  *Replenish and restore food resource: the food resource is restored to its initial state as set by the_environment::food_resource::make() or to a **smaller** abundance. See the_environment::food_resource_replenish_food_items_all()*
- procedure, public destroy => food_resource_destroy_deallocate

  Delete and deallocate food resource object. This class destructor. See the_environment::food_resource_destroy_deall
- procedure, public sort => food_resource_sort_by_size

  Sort the food resource objects within the array by their sizes. The two subroutines below are a variant of the recursive quick-sort algorithm adapted for sorting real components of the the FOOD_RESOURCE object. See the_environment::food_resource_sort_by_size()
- procedure, public reindex => food_resource_reset_iid_all

  Reset individual iid for the food resource. Individual iids must normally coincide with the array order index. If it is changed after sorting, iids no longer reflect the correct index. So this subroutine resets iids to be coinciding with each food item index. See the_environment::food_resource_reset_iid_all()
- procedure, public get_label => food_resource_get_label

  Get the label of the this food resource. See the_environment::food_resource_get_label().
- procedure, public abundance => food_resource_get_abundance

  Get the number of food items in the food resource. See the_environment::food_resource_get_abundance().
- procedure, public location => food_resource_locate_3d

  Get the location object array (array of the_environment::spatial objects) of a food resource object. See the_environment::food_resource_locate_3d()
- procedure, public distance_average => food_resource_calc_average_distance_items

  Calculate the average distance between food items within a resource. See the_environment::food_resource_calc_averag
- procedure, public join => food_resources_collapse

  Collapse several food resources into one. The collapsed resource can then go into the perception system. The properties of the component resources are retained in the collapsed resource. See the_environment::food_resources_collapse()
- procedure, public unjoin => food_resources_update_back

  Transfer back the resulting food resources into their original objects out from a collapsed object from food_↩ resources_collapse. See the_environment::food_resources_update_back()
- procedure, public migrate_vertical => food_resource_migrate_move_items

  Implement vertical migration of all the food items in the resource in a sinusoidal pattern. See the_environment::food_resource
- procedure, public rwalk => food_resource_rwalk_items_default

  Perform a random walk step for all food items within the food resource with default parameters. See the_environment::food_resource_rwalk_items_default().
- procedure, public save_csv => food_resource_save_foods_csv

  Save characteristics of food items in the resource into a CSV file. See the_environment::food_resource_save_foods_csv

Public Attributes

- character(len=label_length) food_label

  Food resource type label.
- integer number_food_items

  Availability of this kind of food, number of food objects that are provided into the environment.
- type(food_item), dimension(:), allocatable food

  Food resource consists of an array of FOOD_ITEM's.

9.24.1 Detailed Description

Definition of the super-type FOOD resource type. This is a superclass, several sub-classes can be defined for different kinds of food and prey objects.
Definition at line 434 of file m_env.f90.

9.24.2 Member Function/Subroutine Documentation

9.24.2.1 make()

procedure, public the_environment::food_resource::make
Make food resource object. This class standard constructor. See the_environment::food_resource_make()
Definition at line 445 of file m_env.f90.

9.24.2.2 replenish()

procedure, public the_environment::food_resource::replenish
Replenish and restore food resource: the food resource is restored to its initial state as set by the_environment::food_resource::make() or to a **smaller** abundance. See the_environment::food_resource_replenish_food_items_all()
Definition at line 450 of file m_env.f90.

9.24.2.3 destroy()

procedure, public the_environment::food_resource::destroy
Delete and deallocate food resource object. This class destructor. See the_environment::food_resource_destroy_dea
Definition at line 453 of file m_env.f90.

9.24.2.4 sort()

procedure, public the_environment::food_resource::sort
Sort the food resource objects within the array by their sizes. The two subroutines below are a variant of the recursive quick-sort algorithm adapted for sorting real components of the the FOOD_RESOURCE object. See the_environment::food_resource_sort_by_size()
Definition at line 459 of file m_env.f90.

9.24.2.5 reindex()

procedure, public the_environment::food_resource::reindex
Reset individual iid for the food resource. Individual iids must normally coincide with the array order index. If it is changed after sorting, iids no longer reflect the correct index. So this subroutine resets iids to be coinciding with each food item index. See the_environment::food_resource_reset_iid_all()
Definition at line 465 of file m_env.f90.

9.24.2.6 get_label()

procedure, public the_environment::food_resource::get_label
Get the label of the this food resource. See the_environment::food_resource_get_label().
Definition at line 468 of file m_env.f90.

9.24.2.7 abundance()

procedure, public the_environment::food_resource::abundance
Get the number of food items in the food resource. See the_environment::food_resource_get_abundance().
Definition at line 471 of file m_env.f90.

9.24.2.8 location()

procedure, public the_environment::food_resource::location
Get the location object array (array of the_environment::spatial objects) of a food resource object. See the_environment::food_resource_locate_3d()
Definition at line 475 of file m_env.f90.

9.24.2.9 distance_average()

`procedure, public the_environment::food_resource::distance_average`
Calculate the average distance between food items within a resource. See the_environment::food_resource_calc_ave
Definition at line 478 of file m_env.f90.

9.24.2.10 join()

`procedure, public the_environment::food_resource::join`
Collapse several food resources into one. The collapsed resource can then go into the perception system. The properties of the component resources are retained in the collapsed resource. See the_environment::food_resources_collapse()
Definition at line 484 of file m_env.f90.

9.24.2.11 unjoin()

`procedure, public the_environment::food_resource::unjoin`
Transfer back the resulting food resources into their original objects out from a collapsed object from food_↩ resources_collapse. See the_environment::food_resources_update_back()
Definition at line 488 of file m_env.f90.

9.24.2.12 migrate_vertical()

`procedure, public the_environment::food_resource::migrate_vertical`
Implement vertical migration of all the food items in the resource in a sinusoidal pattern. See the_environment::food_resour
Definition at line 492 of file m_env.f90.

9.24.2.13 rwalk()

`procedure, public the_environment::food_resource::rwalk`
Perform a random walk step for all food items within the food resource with default parameters. See the_environment::food_resource_rwalk_items_default().
Definition at line 496 of file m_env.f90.

9.24.2.14 save_csv()

`procedure, public the_environment::food_resource::save_csv`
Save characteristics of food items in the resource into a CSV file. See the_environment::food_resource_save_foods_
Definition at line 499 of file m_env.f90.

9.24.3 Member Data Documentation

9.24.3.1 food_label

`character (len=label_length) the_environment::food_resource::food_label`
Food resource type label.
Definition at line 436 of file m_env.f90.

9.24.3.2 number_food_items

`integer the_environment::food_resource::number_food_items`
Availability of this kind of food, number of food objects that are provided into the environment.
Definition at line 439 of file m_env.f90.

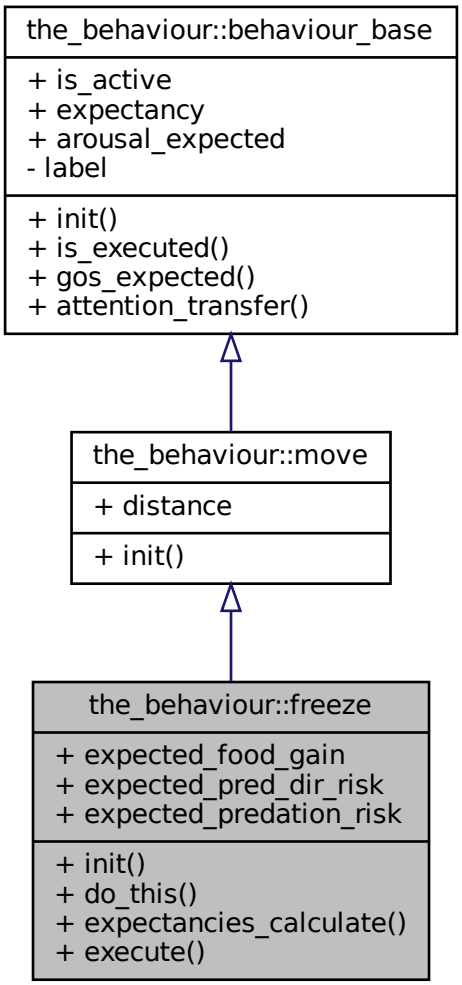
9.24.3.3 food

`type(food_item), dimension(:), allocatable the_environment::food_resource::food`
Food resource consists of an array of `FOOD_ITEM`'s.
Definition at line 441 of file m_env.f90.
The documentation for this type was generated from the following file:

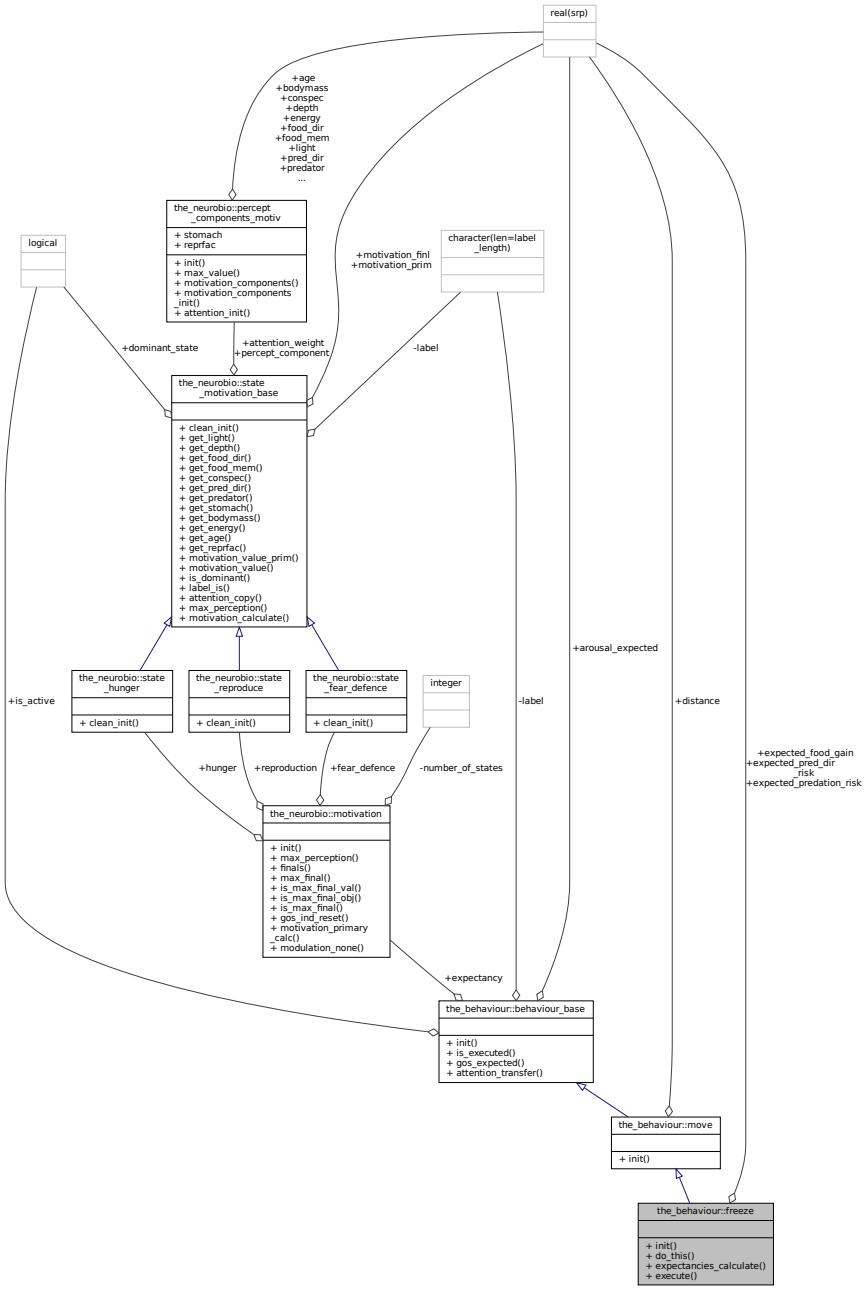
- m_env.f90

9.25 the_behaviour::freeze Type Reference

Freeze is stop any locomotion completely.
Inheritance diagram for the_behaviour::freeze:



Collaboration diagram for the_behaviour::freeze:



Public Member Functions

- procedure, public init => freeze_init_zero

  *Initialise the **freeze** behaviour component to a zero state. Freeze is a special type of move to a zero distance. See* `the_behaviour::freeze_init_zero()`.

- procedure, public do_this => freeze_do_this

  Do freeze by `this_agent` *(the actor agent). Subjective assessment of the motivational value for this is based on the number of food items, conspecifics and predators in the perception object. See* `the_behaviour::freeze_do_this()`.

- procedure, public expectancies_calculate => freeze_motivations_expect

  `the_behaviour::freeze::motivations_expect()` *is a subroutine (re)calculating motivations from fake*

expected perceptions from the procedure `freeze::do_this()` => `the_behaviour::freeze_do_this()`. *See* `the_behaviour:: freeze_motivations_expect()`.

- procedure, public execute => freeze_do_execute

  Execute this behaviour component "freeze" by `this_agent` *agent. See* `the_behaviour::freeze_do_execute()`.

Public Attributes

- real(srp) expected_food_gain

  The expected food gain (body mass increment) is always zero for freezing. Although energy costs are also zero.

- real(srp) expected_pred_dir_risk

  The expected direct predation risk is small and near-zero due to the function the_environment::predator::risk_fish() getting low values with is_freezing=TRUE.

- real(srp) expected_predation_risk

  The expected general predation risk, i.e. the risk depending on the current number of predators in both the perception and memory stack. The expected risk assumes that a freezing predator is not easily noticed by the roaming predators. So the subjective number of predators in the perception is zero in the predation_risk_backend() function.

9.25.1 Detailed Description

Freeze is stop any locomotion completely.
Definition at line 214 of file m_behav.f90.

9.25.2 Member Function/Subroutine Documentation

9.25.2.1 init()

`procedure, public the_behaviour::freeze::init`
Initialise the **freeze** behaviour component to a zero state. Freeze is a special type of move to a zero distance. See `the_behaviour::freeze_init_zero()`.
Definition at line 233 of file m_behav.f90.

9.25.2.2 do_this()

`procedure, public the_behaviour::freeze::do_this`
Do freeze by `this_agent` (the actor agent). Subjective assessment of the motivational value for this is based on the number of food items, conspecifics and predators in the perception object. See `the_behaviour::freeze_do_this()`.
Definition at line 238 of file m_behav.f90.

9.25.2.3 expectancies_calculate()

`procedure, public the_behaviour::freeze::expectancies_calculate`
`the_behaviour::freeze::motivations_expect()` is a subroutine (re)calculating motivations from fake expected perceptions from the procedure `freeze::do_this()` => `the_behaviour::freeze_do_this()`. See `the_behaviour:: freeze_motivations_expect()`.
Definition at line 243 of file m_behav.f90.

9.25.2.4 execute()

`procedure, public the_behaviour::freeze::execute`
Execute this behaviour component "freeze" by `this_agent` agent. See `the_behaviour::freeze_do_execute()`.
Definition at line 246 of file m_behav.f90.

9.25.3 Member Data Documentation

9.25.3.1 expected_food_gain

`real(srp) the_behaviour::freeze::expected_food_gain`
The expected food gain (body mass increment) is always zero for freezing. Although energy costs are also zero.
Definition at line 217 of file m_behav.f90.

9.25.3.2 expected_pred_dir_risk

`real(srp) the_behaviour::freeze::expected_pred_dir_risk`
The expected direct predation risk is small and near-zero due to the function the_environment::predator::risk_fish() getting low values with is_freezing=TRUE.
Definition at line 221 of file m_behav.f90.

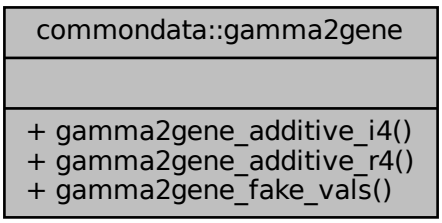
9.25.3.3 expected_predation_risk

`real(srp) the_behaviour::freeze::expected_predation_risk`
The expected general predation risk, i.e. the risk depending on the current number of predators in both the perception and memory stack. The expected risk assumes that a freezing predator is not easily noticed by the roaming predators. So the subjective number of predators in the perception is zero in the predation_risk_backend() function.
Definition at line 228 of file m_behav.f90.
The documentation for this type was generated from the following file:

- m_behav.f90

9.26 commondata::gamma2gene Interface Reference

Sigmoidal relationship between environmental factor and the organism response, as affected by the genotype and environmental error, e.g. perception and neuronal response or intrinsic baseline and phenotypic hormone levels.
Collaboration diagram for commondata::gamma2gene:



Public Member Functions

- real(srp) function gamma2gene_additive_i4 (gs, gh, signal, erpcv)

  The function gamma2gene finds the sigmoid relationship for a complex multicomponent 2-allele impact on the neuronal response.
- real(srp) function gamma2gene_additive_r4 (gs, gh, signal, erpcv)

The function gamma2gene finds the sigmoid relationship for a complex multicomponent 2-allele impact on the neuronal response.

- elemental real(srp) function gamma2gene_fake_vals (signal, gs, gh, n_acomps)

  This "fake" version of the gamma2gene is used to guess the response values in calculations.

9.26.1 Detailed Description

Sigmoidal relationship between environmental factor and the organism response, as affected by the genotype and environmental error, e.g. perception and neuronal response or intrinsic baseline and phenotypic hormone levels.
The real function gamma2gene finds the sigmoid relationship for a multicomponent allele impact on the neuronal response:

$$R = \frac{(P/y_1)^{x_1}}{1+(P/y_1)^{x_1}} + \frac{(P/y_2)^{x_2}}{1+(P/y_2)^{x_2}} + \frac{(P/y_3)^{x_3}}{1+(P/y_3)^{x_3}} ...$$

Here, R is the neuronal response, P the strength of the sensory input (scaled 0-1), and x and y are two genes. The indices refer to the additive components of the alleles. Note that their number is set by the parameter ADDITIVE_COMPS. Further, erpcv defines the coefficient of variation for the perception error (with respect to its true value). *Perception* is calculated as

$$P = \bar{P} + \varepsilon,$$

where $\bar{P}$ is the true environmental variable and ε is Gaussian error. The perception value with error is implemented as a normal Gaussian variate with the mean equal to the true signal value $\bar{P}$ and the coefficient of variation equal to the erpcv input parameter: $erpcv = \frac{\sigma}{\bar{P}}$. Therefore, the raw error variance in the RNORM function is equal to the square: $(erpcv \cdot signal)^2$. We also impose strict *limit* on perception:

$$P > 0.$$

The gamma2gene function is used as a calculation backend in the "umbrella" procedure the_genome::individual_genome: that translates to the_genome::individual_genome::trait_init() and the_genome::individual_genome::

Note

This is the generic interface for gamma2gene, currently there are two version making use of the additive allele components (normally used) accepting allele values either integer (assumed raw alleles) or real (assumed values rescaled to 0:1) type.

The function involving additive allele components has two variants: If called with *integer* parameters one and two (gs and gh) it automatically invokes the allelescale function to convert integer allele values to real raw values within the range 1:0 that go to the sigmoid function (via alleleconv). However, when these arguments are *real* type, it is assumed that it is the true converted 0:1 real gene values and allelescale is *not* invoked prior to alleleconv.

It would be more economical to use a single allele conversion function that does both allelescale and alleleconv. It is here for compatibility with the earlier model. Also, this might be a little more intuitive.

Definition at line 5269 of file m_common.f90.

9.26.2 Member Function/Subroutine Documentation

9.26.2.1 gamma2gene_additive_i4()

```
real(srp) function commondata::gamma2gene::gamma2gene_additive_i4 (
            integer, dimension(:), intent(in) gs,
            integer, dimension(:), intent(in) gh,
            real(srp), intent(in) signal,
            real(srp), intent(in), optional erpcv )
```

The function gamma2gene finds the sigmoid relationship for a complex multicomponent 2-allele impact on the neuronal response.
The real function gamma2gene finds the sigmoid relationship for a multicomponent allele impact on the neuronal response:

$$R = \frac{(P/y_1)^{x_1}}{1+(P/y_1)^{x_1}} + \frac{(P/y_2)^{x_2}}{1+(P/y_2)^{x_2}} + \frac{(P/y_3)^{x_3}}{1+(P/y_3)^{x_3}} ...$$

Here, R is the neuronal response, P the strength of the sensory input (scaled 0-1), and x and y are two genes. The indices refer to the additive components of the alleles. Note that their number is set by the parameter ADDITIVE↩ _COMPS. Further, erpcv defines the coefficient of variation for the perception error (with respect to its true value).

Returns

returns the neuronal response.

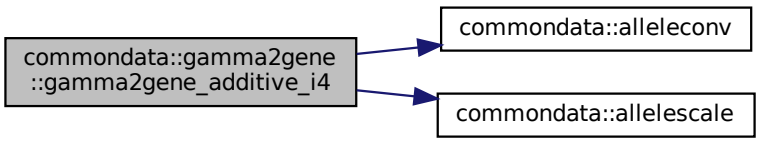
Parameters

in	*gs*	shape: Gene/constant determining the shape of the gamma function. Note that the raw integer gene values are accepted by this function as commondata::allelescale() is called automatically inside.
in	*gh*	half-max effect: Gene/constant for the signal strength giving half max effect. Note that the raw integer gene values are accepted by this function as commondata::allelescale() is called automatically inside.
in	*signal*	perception: Input value of (external or internal) stimulus perception.
in	*erpcv*	error: Additive error of stimulus perception, Gaussian variance added to the true environmental variable. If this parameter is absent, no perception error is introduced. Maxima function for quick calc: `g2gene(p,x,y,n) := n * ( (p/y)^x / (1+(p/y)^x) );`

Warning

This version of gamma2gene accepts *integer* arrays. It *does* invoke commondata::allelescale() automatically inside.

Definition at line 7003 of file m_common.f90.
Here is the call graph for this function:



9.26.2.2 gamma2gene_additive_r4()

```
real(srp) function commondata::gamma2gene::gamma2gene_additive_r4 (
             real(srp), dimension(:), intent(in) gs,
             real(srp), dimension(:), intent(in) gh,
             real(srp), intent(in) signal,
             real(srp), intent(in), optional erpcv )
```

The function gamma2gene finds the sigmoid relationship for a complex multicomponent 2-allele impact on the neuronal response.

Returns

returns the neuronal response.

Parameters

in	*shape*	Gene/constant determining the shape of the gamma function. Note that the raw integer gene values are accepted by this function as commondata::allelescale() is called automatically inside.
in	*half-max*	effect: Gene/constant for the signal strength giving half max effect. Note that the raw integer gene values are accepted by this function as commondata::allelescale() is called automatically inside.
in	*perception*	Input value of (external or internal) stimulus perception.
in	*error*	Additive error of stimulus perception, Gaussian variance added to the true environmental variable. If this parameter is absent, no perception error is introduced.

The real function gamma2gene finds the sigmoid relationship for a multicomponent allele impact on the neuronal response:

$$R = \frac{(P/y_1)^{x_1}}{1+(P/y_1)^{x_1}} + \frac{(P/y_2)^{x_2}}{1+(P/y_2)^{x_2}} + \frac{(P/y_3)^{x_3}}{1+(P/y_3)^{x_3}} \ldots$$

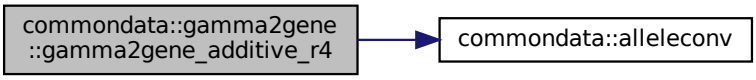
Here, R is the neuronal response, P the strength of the sensory input (scaled 0-1), and x and y are two genes. The indices refer to the additive components of the alleles. Note that their number is set by the parameter ADDITIVE_COMPS. Further, erpcv defines the coefficient of variation for the perception error (with respect to its true value).
Maxima function for quick calc:

```
g2gene(p,x,y,n) := n * ( (p/y)^x / (1+(p/y)^x) );
```

Warning

This version of gamma2gene accepts *real* arrays. It does *not* invoke commondata::allelescale() automatically inside.

Definition at line 7115 of file m_common.f90.
Here is the call graph for this function:



9.26.2.3 gamma2gene_fake_vals()

```
elemental real(srp) function commondata::gamma2gene::gamma2gene_fake_vals (
            real(srp), intent(in) signal,
            real(srp), intent(in), optional gs,
            real(srp), intent(in), optional gh,
            integer, intent(in), optional n_acomps )
```

This "fake" version of the gamma2gene is used to guess the response values in calculations.

Return values

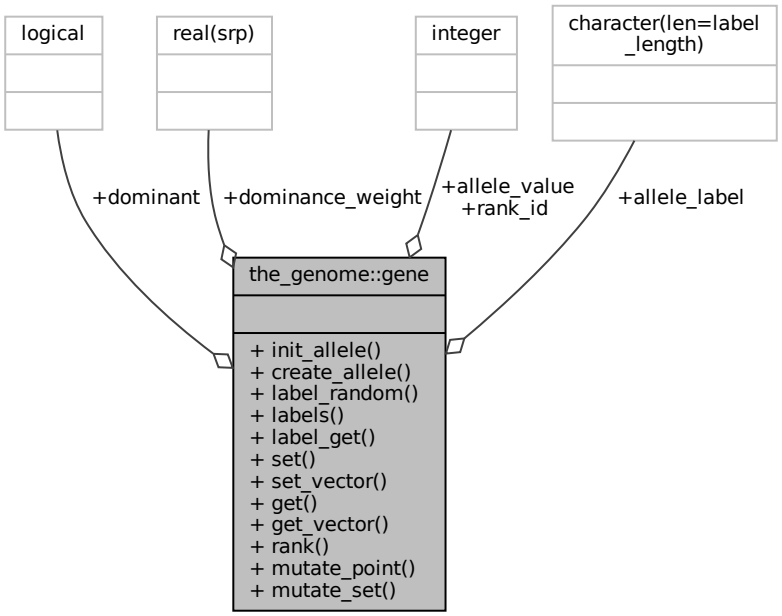
predicted_val	a predicted value (scalar or array) of the sigmoidal neuronal response function. See gamma2gene for details.

Definition at line 7193 of file m_common.f90.
The documentation for this interface was generated from the following file:

- m_common.f90

9.27 the_genome::gene Type Reference

This describes an individual gene object. See the genome structure for as general description and gene for details.
Collaboration diagram for the_genome::gene:



Public Member Functions

- procedure, public init_allele => allele_init_random

  init alleles with random values, labels not set here, use this function for startup initialisations of random agents See `the_genome::allele_init_random()`
- procedure, public create_allele => allele_create_zero

  create empty zero allele object, should be used for offspring inits as we do not need to init them with random values, they will get them from the parents using inherit function set See `the_genome::allele_create_zero()`
- procedure, public label_random => allele_label_init_random

  init label alleles random See `the_genome::allele_label_init_random()`
- procedure, public labels => allele_label_set

  set labels for the allele See `the_genome::allele_label_set()`
- procedure, public label_get => allele_label_get

  get the allele label See `the_genome::allele_label_get()`
- procedure, public set => allele_value_set

  set individual value of allele See `the_genome::allele_value_set()`
- procedure, public set_vector => alleles_value_vector_set

  set the vector of additive allele components See `the_genome::alleles_value_vector_set()`
- procedure, public get => allele_value_get

  get the value of the allele See `the_genome::allele_value_get()`
- procedure, public get_vector => allele_values_vector_get

  get the vector of additive allele components See `the_genome::allele_values_vector_get()`

- procedure, public rank => allele_rank_id_set

  set rank_id for the allele See the_genome::allele_rank_id_set()
- procedure, public mutate_point => allele_mutate_random

  Introduce a random point mutation to one (random) of the alleles See the_genome::allele_mutate_random()
- procedure, public mutate_set => allele_mutate_random_batch

  Introduce random mutations to the whole allele components set See the_genome::allele_mutate_random_batch()

Public Attributes

- character(len=label_length) allele_label

  sets a descriptive label of the allele, e.g its role and purpose
- integer, dimension(additive_comps) allele_value

  Sets the value of the allele that is stored and evolved.
- logical dominant

  sets if the allele is dominant
- real(srp) dominance_weight

  sets the multiplicative dominance weight
- integer rank_id

  rank_id of the gene, needed for sorting alleles within the chromosome

9.27.1 Detailed Description

This describes an individual gene object. See the genome structure for as general description and gene for details.
Definition at line 34 of file m_genome.f90.

9.27.2 Member Function/Subroutine Documentation

9.27.2.1 init_allele()

`procedure, public the_genome::gene::init_allele`
init alleles with random values, labels not set here, use this function for startup initialisations of random agents See the_genome::allele_init_random()
Definition at line 65 of file m_genome.f90.

9.27.2.2 create_allele()

`procedure, public the_genome::gene::create_allele`
create empty zero allele object, should be used for offspring inits as we do not need to init them with random values, they will get them from the parents using inherit function set See the_genome::allele_create_zero()
Definition at line 70 of file m_genome.f90.

9.27.2.3 label_random()

`procedure, public the_genome::gene::label_random`
init label alleles random See the_genome::allele_label_init_random()
Definition at line 73 of file m_genome.f90.

9.27.2.4 labels()

`procedure, public the_genome::gene::labels`
set labels for the allele See the_genome::allele_label_set()
Definition at line 76 of file m_genome.f90.

9.27.2.5 label_get()

procedure, public the_genome::gene::label_get
get the allele label See the_genome::allele_label_get()
Definition at line 79 of file m_genome.f90.

9.27.2.6 set()

procedure, public the_genome::gene::set
set individual value of allele See the_genome::allele_value_set()
Definition at line 82 of file m_genome.f90.

9.27.2.7 set_vector()

procedure, public the_genome::gene::set_vector
set the vector of additive allele components See the_genome::alleles_value_vector_set()
Definition at line 85 of file m_genome.f90.

9.27.2.8 get()

procedure, public the_genome::gene::get
get the value of the allele See the_genome::allele_value_get()
Definition at line 88 of file m_genome.f90.

9.27.2.9 get_vector()

procedure, public the_genome::gene::get_vector
get the vector of additive allele components See the_genome::allele_values_vector_get()
Definition at line 91 of file m_genome.f90.

9.27.2.10 rank()

procedure, public the_genome::gene::rank
set rank_id for the allele See the_genome::allele_rank_id_set()
Definition at line 94 of file m_genome.f90.

9.27.2.11 mutate_point()

procedure, public the_genome::gene::mutate_point
Introduce a random point mutation to one (random) of the alleles See the_genome::allele_mutate_random()
Definition at line 97 of file m_genome.f90.

9.27.2.12 mutate_set()

procedure, public the_genome::gene::mutate_set
Introduce random mutations to the whole allele components set See the_genome::allele_mutate_random_batch()
Definition at line 100 of file m_genome.f90.

9.27.3 Member Data Documentation

9.27.3.1 allele_label

`character(len=label_length) the_genome::gene::allele_label`
sets a descriptive label of the allele, e.g its role and purpose
Definition at line 36 of file m_genome.f90.

9.27.3.2 allele_value

`integer, dimension(additive_comps) the_genome::gene::allele_value`
Sets the value of the allele that is stored and evolved.

Note

In the new version allele values are *INTEGER* rather than *REAL*. Integer genome is not affected by the CPU precision and does not suffer from FPU rounding errors. This is what is expected from the genome: genes should be atomic, fixed, and never subject to any uncontrollable fluctuations and drift. Otherwise no "inheritance" is guaranteed. Only controlled mutations are allowed. Integer calculations will also have higher calculation speed and may hopefully avoid IEEE float point errors (overflow/underflow). Also, we may in future use more realistic limited-range allele functions to mimic real DNA structure. If we have sufficiently large range of possible allele values, e.g. 1:10000 and integer-to-real conversion function for converting these true integer allele values to real values within 0.:1. in the gamma neural response function, this would not have a much different effect compared with the old real-value gene implementation.

Definition at line 54 of file m_genome.f90.

9.27.3.3 dominant

`logical the_genome::gene::dominant`
sets if the allele is dominant
Definition at line 56 of file m_genome.f90.

9.27.3.4 dominance_weight

`real(srp) the_genome::gene::dominance_weight`
sets the multiplicative dominance weight
Definition at line 58 of file m_genome.f90.

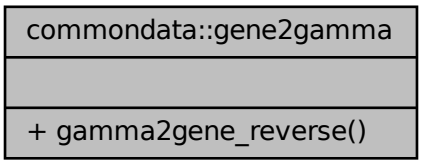
9.27.3.5 rank_id

`integer the_genome::gene::rank_id`
rank_id of the gene, needed for sorting alleles within the chromosome
Definition at line 60 of file m_genome.f90.
The documentation for this type was generated from the following file:

- m_genome.f90

9.28 commondata::gene2gamma Interface Reference

Collaboration diagram for commondata::gene2gamma:



Public Member Functions

- elemental real(srp) function gamma2gene_reverse (neuronal_response, gs, gh, nc)

  Reverse-calculate perception value from the given neural response value.

9.28.1 Detailed Description

Definition at line 5278 of file m_common.f90.

9.28.2 Member Function/Subroutine Documentation

9.28.2.1 gamma2gene_reverse()

```
elemental real(srp) function commondata::gene2gamma::gamma2gene_reverse (
            real(srp), intent(in) neuronal_response,
            real(srp), intent(in) gs,
            real(srp), intent(in) gh,
            integer, intent(in), optional nc )
```

Reverse-calculate perception value from the given neural response value.
Calculates the value of the raw perception from the neural response function. This is the reverse of the gamma2gene with many components. It is assumed that all x and y values are the same, so the equation solved for the most trivial case. Calculated according to the formula:

$$P = y\left(\frac{R}{n-R}\right)^{1/x},$$

where P is the perception value, R is the neural response, x and y are two genes.

Returns

Signal level for specific neural response, back calculated.

Parameters

in	*neuronal_response*	neuronal response.
in	*gs*	shape parameter of the sigmoid function.
in	*gh*	half-max parameter of the sigmoid function.
in	*nc*	Number of additive components. Optional, if absent assumed 1 (single component).

Note

This function is useful for guessing the average start values of genetically determined traits with Gaussian distribution.

Note that it is quite difficult to get really small `gamma2gene` values as the signal value should be really small: e.g. to get neural response 1.5E-5 (Fulton condition), we need signal = 2E-12. So, the function very quickly loses precision as we approach really low values. Need kind 8 or 16 precision?

Warning

This is quite a crude guess at low values. At lower values *underestimates* R, real value is higher. This is due to the limitation that R should never be below zero, causing above-zero truncation.

Returns

Signal level for specific neural response, back calculated.

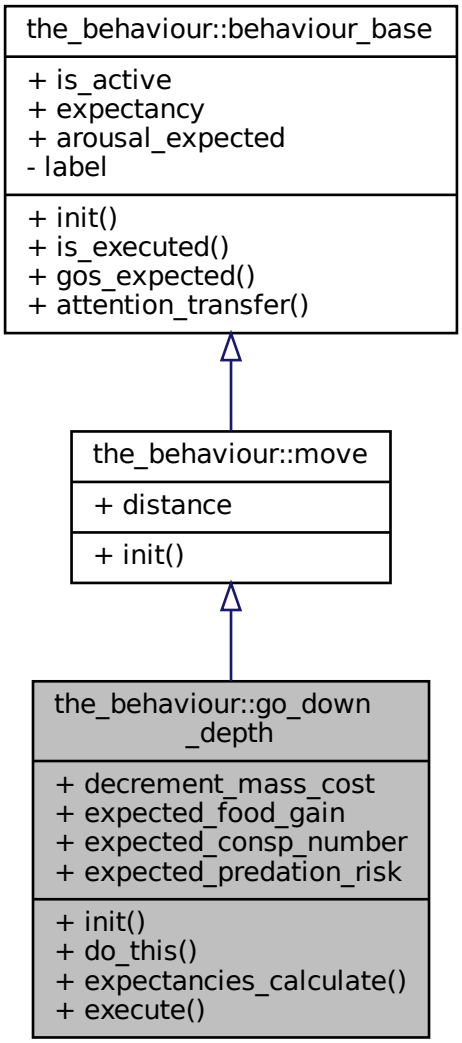
Definition at line 7256 of file m_common.f90.
The documentation for this interface was generated from the following file:

- m_common.f90

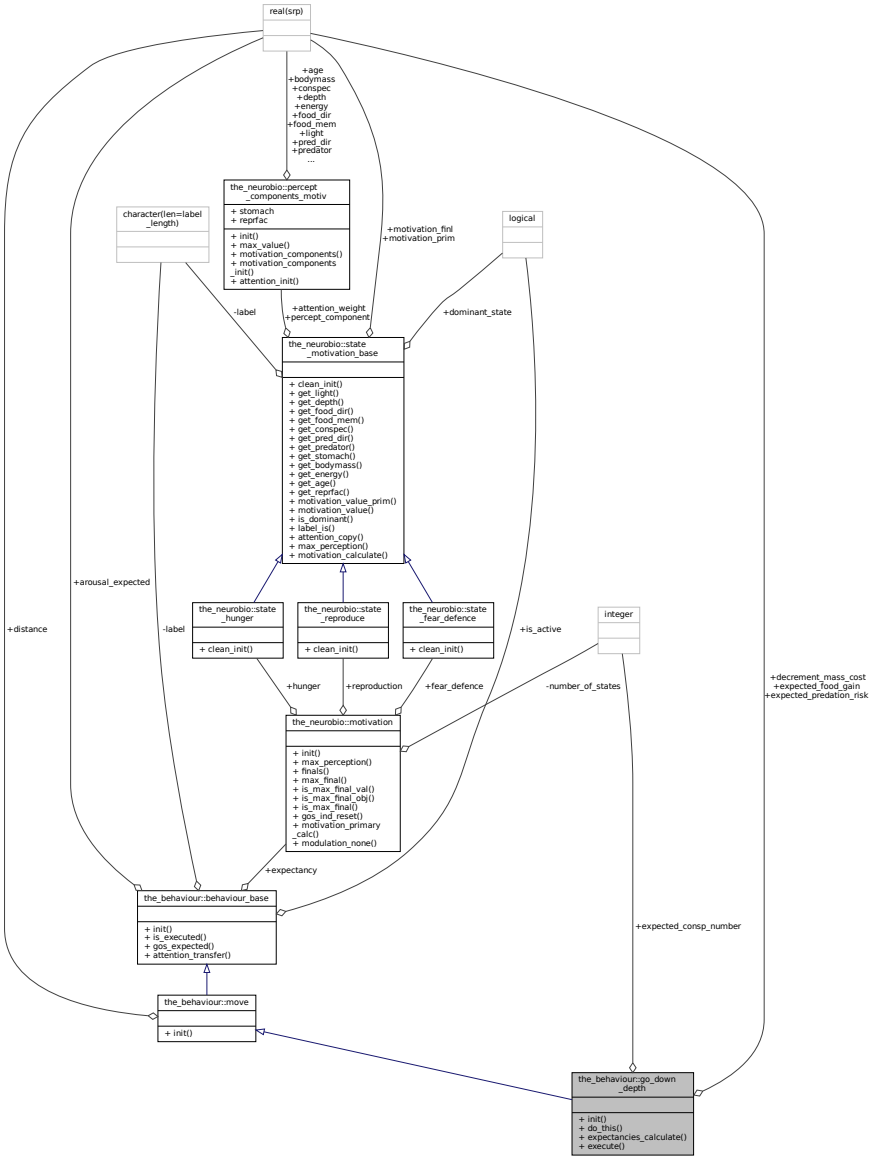
9.29 the_behaviour::go_down_depth Type Reference

Go down dive deeper.

Inheritance diagram for the_behaviour::go_down_depth:



Collaboration diagram for the_behaviour::go_down_depth:



Public Member Functions

- procedure, public init => go_down_depth_init_zero

  *Initialise the **go down to a deeper spatial layer** behaviour component to a zero state. See* `the_behaviour::go_down_depth_in`
- procedure, public do_this => go_down_do_this

  `do_this` *performs the agent's action without changing the agent or the environment.*
- procedure, public expectancies_calculate => go_down_motivations_expect

  `expectancies_calculate` *is a subroutine (re)calculating motivations from fake expected perceptions following from* `do_this`.
- procedure, public execute => go_down_do_execute

  Execute this behaviour component "go down" by `this_agent` *agent. See* `the_behaviour::go_down_do_execute()`.

Public Attributes

- real(srp) decrement_mass_cost

The cost of the swimming downwards. Should be relatively low, much smaller than the cost of active locomotion to the same distance (in terms of the body length as set by condition_cost_swimming_burst()). This is because it is assumed to be based on the hydrodynamic (swimbladder) volume manipulation rather than active propulsion.

- real(srp) expected_food_gain

  The expected food gain (body mass increment) from the food items deeper than the actor agent. This value is based on the number and average mass of food items below the agent's current horizon.

- integer expected_consp_number

  The expected number of conspecifics at the layer below. This value is based on the number of conspecifics below the agent's current horizon.

- real(srp) expected_predation_risk

  The expected predation risk at the layer below. This value is based on the number of predators below the agent's current horizon.

9.29.1 Detailed Description

Go down dive deeper.
Definition at line 394 of file m_behav.f90.

9.29.2 Member Function/Subroutine Documentation

9.29.2.1 init()

`procedure, public the_behaviour::go_down_depth::init`
Initialise the **go down to a deeper spatial layer** behaviour component to a zero state. See `the_behaviour::go_down_depth_`
Definition at line 415 of file m_behav.f90.

9.29.2.2 do_this()

`procedure, public the_behaviour::go_down_depth::do_this`
`do_this` performs the agent's action without changing the agent or the environment.
Definition at line 418 of file m_behav.f90.

9.29.2.3 expectancies_calculate()

`procedure, public the_behaviour::go_down_depth::expectancies_calculate`
`expectancies_calculate` is a subroutine (re)calculating motivations from fake expected perceptions following from `do_this`.

Note

Note that this is the computational engine to assess the expected GOS of the behaviour, it is called from within the base root behaviour class-bound polymorphic `gos_expect`. See `the_behaviour::go_down_motivations_exp`

Definition at line 425 of file m_behav.f90.

9.29.2.4 execute()

`procedure, public the_behaviour::go_down_depth::execute`
Execute this behaviour component "go down" by `this_agent` agent. See `the_behaviour::go_down_do_execute()`.
Definition at line 428 of file m_behav.f90.

9.29.3 Member Data Documentation

9.29.3.1 decrement_mass_cost

`real(srp) the_behaviour::go_down_depth::decrement_mass_cost`
The cost of the swimming downwards. Should be relatively low, much smaller than the cost of active locomotion to the same distance (in terms of the body length as set by condition_cost_swimming_burst()). This is because it is assumed to be based on the hydrodynamic (swimbladder) volume manipulation rather than active propulsion.
Definition at line 400 of file m_behav.f90.

9.29.3.2 expected_food_gain

`real(srp) the_behaviour::go_down_depth::expected_food_gain`
The expected food gain (body mass increment) from the food items deeper than the actor agent. This value is based on the number and average mass of food items below the agent's current horizon.
Definition at line 404 of file m_behav.f90.

9.29.3.3 expected_consp_number

`integer the_behaviour::go_down_depth::expected_consp_number`
The expected number of conspecifics at the layer below. This value is based on the number of conspecifics below the agent's current horizon.
Definition at line 407 of file m_behav.f90.

9.29.3.4 expected_predation_risk

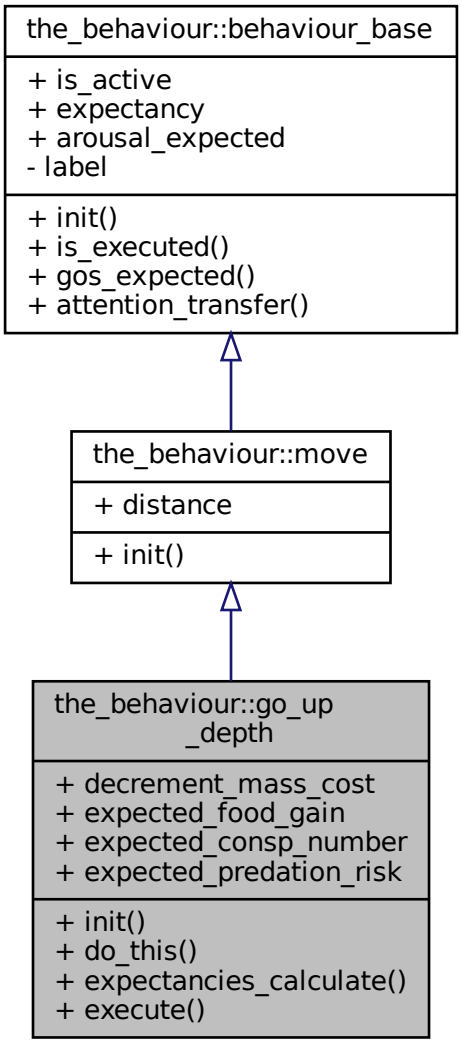
`real(srp) the_behaviour::go_down_depth::expected_predation_risk`
The expected predation risk at the layer below. This value is based on the number of predators below the agent's current horizon.
Definition at line 410 of file m_behav.f90.
The documentation for this type was generated from the following file:

- m_behav.f90

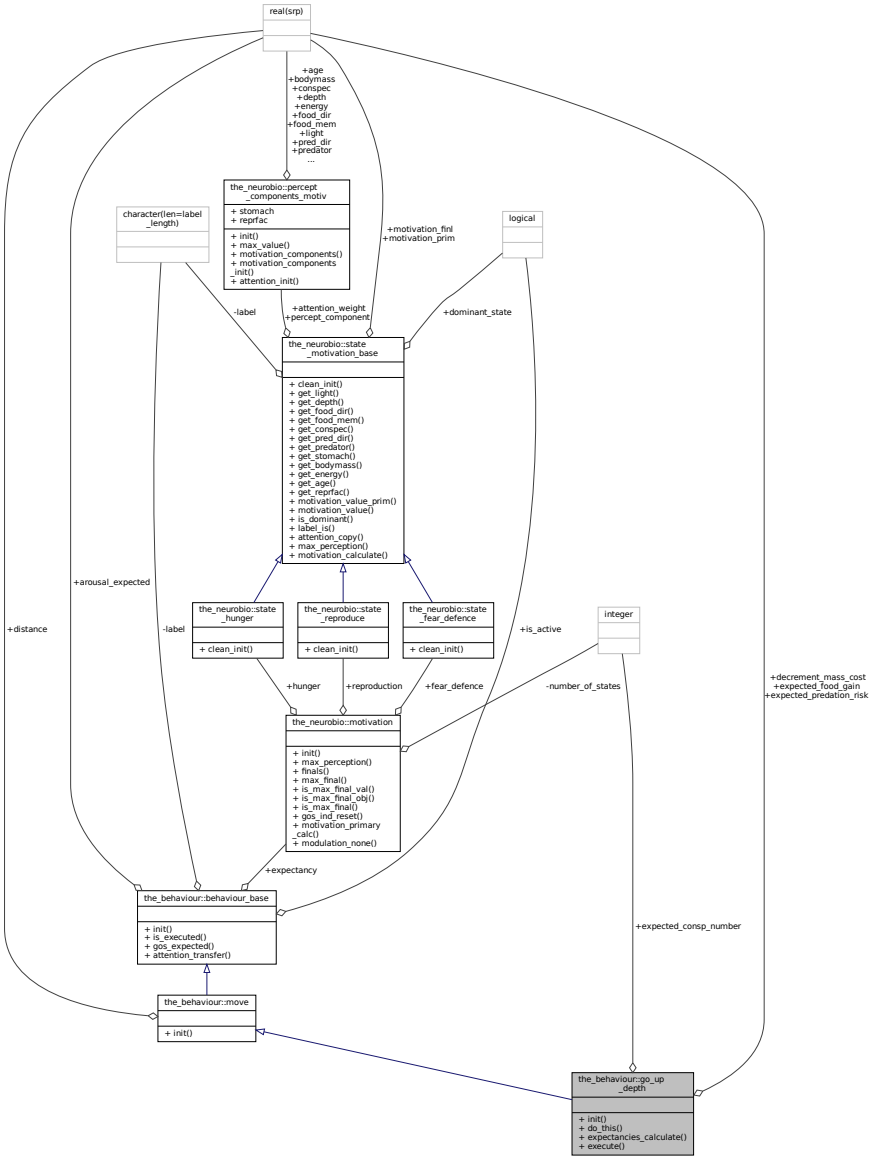
9.30 the_behaviour::go_up_depth Type Reference

Go up raise to a smaller depth. TODO: abstract type linking both Up and Down.

Inheritance diagram for the_behaviour::go_up_depth:



Collaboration diagram for the_behaviour::go_up_depth:



Public Member Functions

- procedure, public init => go_up_depth_init_zero

  *Initialise the **go up to a shallower spatial layer** behaviour component to a zero state. See* `the_behaviour::go_up_depth_init`

- procedure, public do_this => go_up_do_this

  `do_this` *performs the agent's action without changing the agent or the environment. See* `the_behaviour::go_up_do_this()`.

- procedure, public expectancies_calculate => go_up_motivations_expect

  `expectancies_calculate` *is a subroutine (re)calculating motivations from fake expected perceptions following from* `do_this`.

- procedure, public execute => go_up_do_execute

  Execute this behaviour component "go up" by `this_agent` *agent. See* `the_behaviour::go_up_do_execute()`.

Public Attributes

- real(srp) decrement_mass_cost

The cost of the swimming downwards. Should be relatively low, much smaller than the cost of active locomotion to the same distance (in terms of the body length as set by condition_cost_swimming_burst()). This is because it is assumed to be based on the hydrodynamic (swimbladder) volume manipulation rather than active propulsion.

- real(srp) expected_food_gain

  The expected food gain (body mass increment) from the food items deeper than the actor agent. This value is based on the number and average mass of food items below the agent's current horizon.

- integer expected_consp_number

  The expected number of conspecifics at the layer below. This value is based on the number of conspecifics below the agent's current horizon.

- real(srp) expected_predation_risk

  The expected predation risk at the layer below. This value is based on the number of predators below the agent's current horizon.

9.30.1 Detailed Description

Go up raise to a smaller depth. TODO: abstract type linking both Up and Down.
Definition at line 433 of file m_behav.f90.

9.30.2 Member Function/Subroutine Documentation

9.30.2.1 init()

`procedure, public the_behaviour::go_up_depth::init`
Initialise the **go up to a shallower spatial layer** behaviour component to a zero state. See `the_behaviour::go_up_depth_in`
Definition at line 454 of file m_behav.f90.

9.30.2.2 do_this()

`procedure, public the_behaviour::go_up_depth::do_this`
`do_this` performs the agent's action without changing the agent or the environment. See `the_behaviour::go_up_do_this`
Definition at line 458 of file m_behav.f90.

9.30.2.3 expectancies_calculate()

`procedure, public the_behaviour::go_up_depth::expectancies_calculate`
`expectancies_calculate` is a subroutine (re)calculating motivations from fake expected perceptions following from `do_this`.

Note

Note that this is the computational engine to assess the expected GOS of the behaviour, it is called from within the base root behaviour class-bound polymorphic `gos_expect`. See `the_behaviour::go_up_motivations_expec`

Definition at line 465 of file m_behav.f90.

9.30.2.4 execute()

`procedure, public the_behaviour::go_up_depth::execute`
Execute this behaviour component "go up" by `this_agent` agent. See `the_behaviour::go_up_do_execute()`.
Definition at line 468 of file m_behav.f90.

9.30.3 Member Data Documentation

9.30.3.1 decrement_mass_cost

`real(srp) the_behaviour::go_up_depth::decrement_mass_cost`
The cost of the swimming downwards. Should be relatively low, much smaller than the cost of active locomotion to the same distance (in terms of the body length as set by condition_cost_swimming_burst()). This is because it is assumed to be based on the hydrodynamic (swimbladder) volume manipulation rather than active propulsion.
Definition at line 439 of file m_behav.f90.

9.30.3.2 expected_food_gain

`real(srp) the_behaviour::go_up_depth::expected_food_gain`
The expected food gain (body mass increment) from the food items deeper than the actor agent. This value is based on the number and average mass of food items below the agent's current horizon.
Definition at line 443 of file m_behav.f90.

9.30.3.3 expected_consp_number

`integer the_behaviour::go_up_depth::expected_consp_number`
The expected number of conspecifics at the layer below. This value is based on the number of conspecifics below the agent's current horizon.
Definition at line 446 of file m_behav.f90.

9.30.3.4 expected_predation_risk

`real(srp) the_behaviour::go_up_depth::expected_predation_risk`
The expected predation risk at the layer below. This value is based on the number of predators below the agent's current horizon.
Definition at line 449 of file m_behav.f90.
The documentation for this type was generated from the following file:

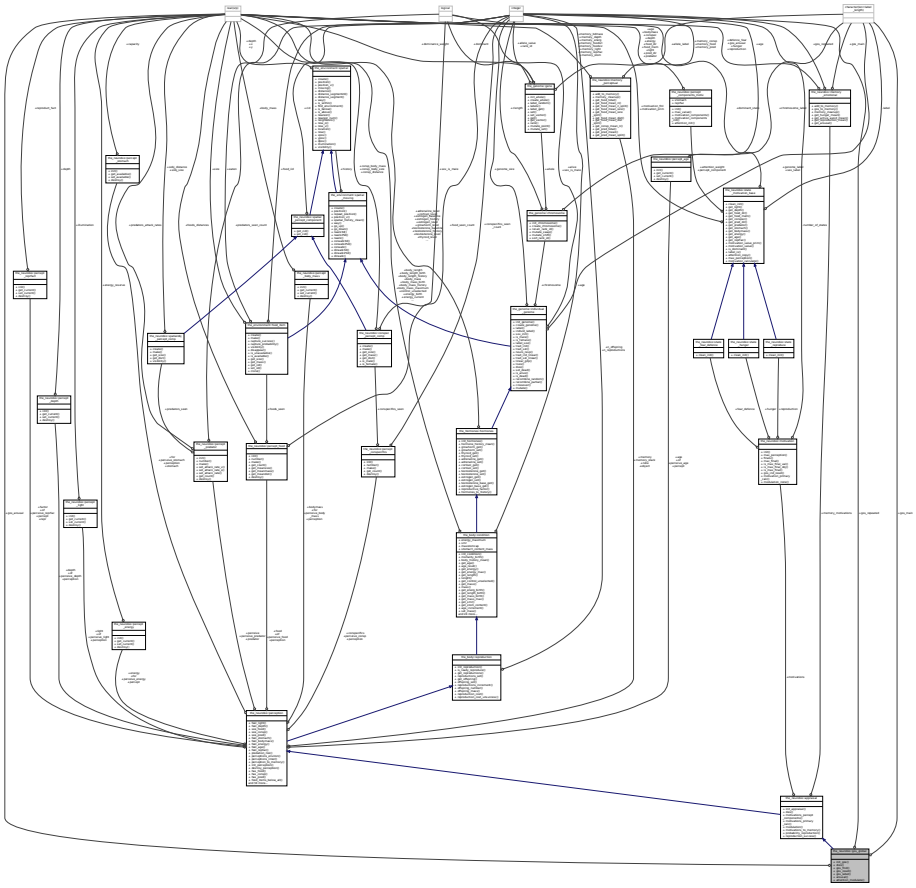
- m_behav.f90

9.31 the_neurobio::gos_global Type Reference

Global organismic state (GOS) level. GOS is defined by the dominant motivational state component (STATE↩_), namely, by the logical flag %dominant_state. If this logical flag is TRUE for a particular motivational state component, this state is the GOS. Thus, there should be is no separate data component(s) e.g. "value" for GOS. The values the_neurobio::gos_global::gos_main and the_neurobio::gos_global::gos_arousal can be inferred from the motivations, here are doubled mainly for convenience. See "From perception to GOS" for an overview.

Inheritance diagram for the_neurobio::gos_global:

Collaboration diagram for the_neurobio::gos_global:



Public Member Functions

- procedure, public init_gos => gos_init_zero_state

  Initialise GOS engine components to a zero state. See `the_neurobio::gos_init_zero_state()`.
- procedure, public dies => gos_agent_set_dead

  Set the individual to be ***dead****. This method overrides the the_genome::individual_genome::dies() method, nullifying all reproductive and neurobiological and behavioural objects. However, this function does not deallocate the individual agent object, this may be a separate destructor function. The* `dies` *method is implemented at the following levels of the agent object hierarchy (upper overrides the lower level):*
- procedure, public gos_find => gos_find_global_state

  Find and set the global organismic state (GOS) based on the various available motivation values. See `the_neurobio::gos_find_global_state()`.
- procedure, public gos_reset => gos_reset_motivations_non_dominant

  Reset all motivation states as NOT dominant with respect to the GOS. See `the_neurobio::gos_reset_motivations_non_do`
- procedure, public gos_label => gos_global_get_label

  Get the current global organismic state (GOS). See `the_neurobio::gos_global_get_label()`.
- procedure, public arousal => gos_get_arousal_level

  Get the overall level of arousal. Arousal is the current level of the dominant motivation that has brought about the current GOS at the previous time step. See `the_neurobio::gos_get_arousal_level()`.
- procedure, public attention_modulate => gos_attention_modulate_weights

  Modulate the attention weights to suppress all perceptions alternative to the current GOS. This is done using the attention modulation interpolation curve. See `the_neurobio::gos_attention_modulate_weights()`.

Public Attributes

- character(len=label_length) gos_main

  Current global organismic state (GOS). Obtained from the GOS-specific emotional state %label data component.
- real(srp) gos_arousal

  This is the current value of the dominant motivation.
- integer gos_repeated

  Integer number of the same GOS repetition, e.g. if GOS is the same the second time, gets 2 etc. Needed to asymptotically reduce GOS arousal when it is repeated, so smaller stimuli could overtake control.

9.31.1 Detailed Description

Global organismic state (GOS) level. GOS is defined by the dominant motivational state component (`STATE_`), namely, by the logical flag %dominant_state. If this logical flag is TRUE for a particular motivational state component, this state is the GOS. Thus, there should be is no separate data component(s) e.g. "value" for GOS. The values the_neurobio::gos_global::gos_main and the_neurobio::gos_global::gos_arousal can be inferred from the motivations, here are doubled mainly for convenience. See "From perception to GOS" for an overview.
Definition at line 1304 of file m_neuro.f90.

9.31.2 Member Function/Subroutine Documentation

9.31.2.1 init_gos()

`procedure, public the_neurobio::gos_global::init_gos`
Initialise GOS engine components to a zero state. See `the_neurobio::gos_init_zero_state()`.
Definition at line 1317 of file m_neuro.f90.

9.31.2.2 dies()

`procedure, public the_neurobio::gos_global::dies`
Set the individual to be **dead**. This method overrides the the_genome::individual_genome::dies() method, nullifying all reproductive and neurobiological and behavioural objects. However, this function does not deallocate the individual agent object, this may be a separate destructor function. The `dies` method is implemented at the following levels of the agent object hierarchy (upper overrides the lower level):

- the_genome::individual_genome::dies();
- the_neurobio::appraisal::dies();
- the_neurobio::gos_global::dies();
- the_individual::individual_agent::dies().

See `the_individual::gos_agent_set_dead()`.
Definition at line 1331 of file m_neuro.f90.

9.31.2.3 gos_find()

`procedure, public the_neurobio::gos_global::gos_find`
Find and set the global organismic state (GOS) based on the various available motivation values. See `the_neurobio::gos_find_global_state()`.
Definition at line 1335 of file m_neuro.f90.

9.31.2.4 gos_reset()

`procedure, public the_neurobio::gos_global::gos_reset`
Reset all motivation states as NOT dominant with respect to the GOS. See `the_neurobio::gos_reset_motivations_non`
Definition at line 1338 of file m_neuro.f90.

9.31.2.5 gos_label()

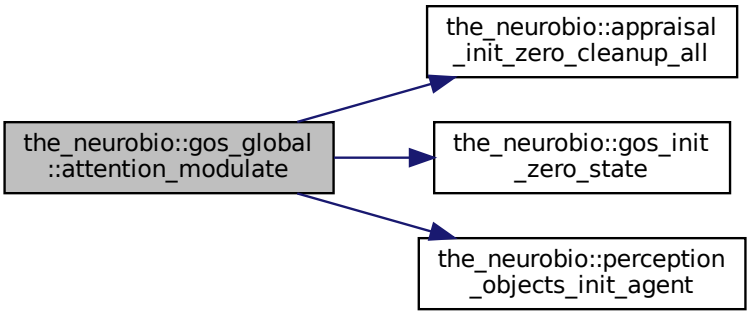
`procedure, public the_neurobio::gos_global::gos_label`
Get the current global organismic state (GOS). See `the_neurobio::gos_global_get_label()`.
Definition at line 1341 of file m_neuro.f90.

9.31.2.6 arousal()

`procedure, public the_neurobio::gos_global::arousal`
Get the overall level of arousal. Arousal is the current level of the dominant motivation that has brought about the current GOS at the previous time step. See `the_neurobio::gos_get_arousal_level()`.
Definition at line 1346 of file m_neuro.f90.

9.31.2.7 attention_modulate()

`procedure, public the_neurobio::gos_global::attention_modulate`
Modulate the attention weights to suppress all perceptions alternative to the current GOS. This is done using the attention modulation interpolation curve. See `the_neurobio::gos_attention_modulate_weights()`.
Definition at line 1351 of file m_neuro.f90.
Here is the call graph for this function:



9.31.3 Member Data Documentation

9.31.3.1 gos_main

`character(len=label_length) the_neurobio::gos_global::gos_main`
Current global organismic state (GOS). Obtained from the GOS-specific emotional state %label data component.
Definition at line 1307 of file m_neuro.f90.

9.31.3.2 gos_arousal

`real(srp) the_neurobio::gos_global::gos_arousal`
This is the current value of the dominant motivation.
Definition at line 1309 of file m_neuro.f90.

9.31.3.3 gos_repeated

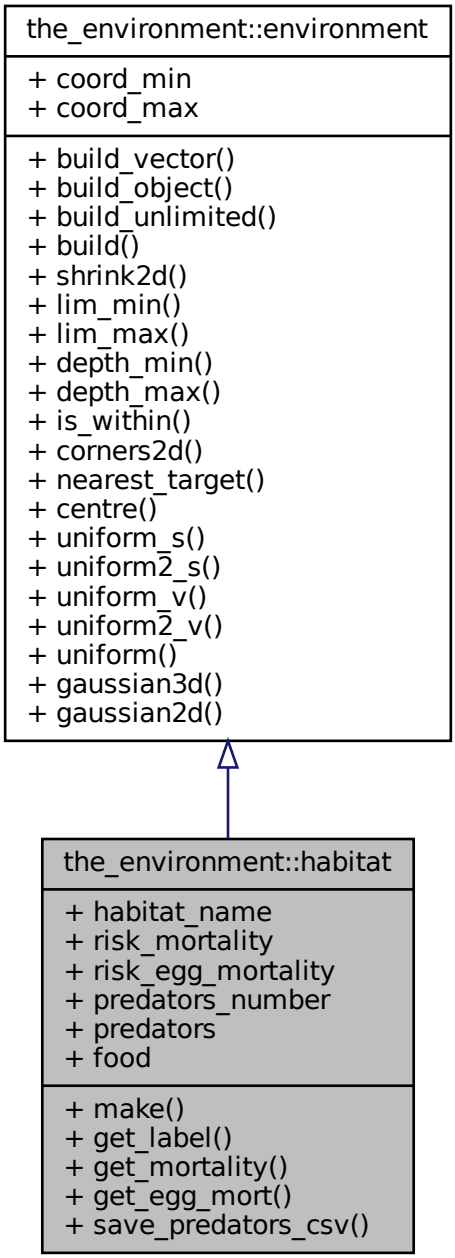
`integer the_neurobio::gos_global::gos_repeated`
Integer number of the same GOS repetition, e.g. if GOS is the same the second time, gets 2 etc. Needed to asymptotically reduce GOS arousal when it is repeated, so smaller stimuli could overtake control.
Definition at line 1313 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

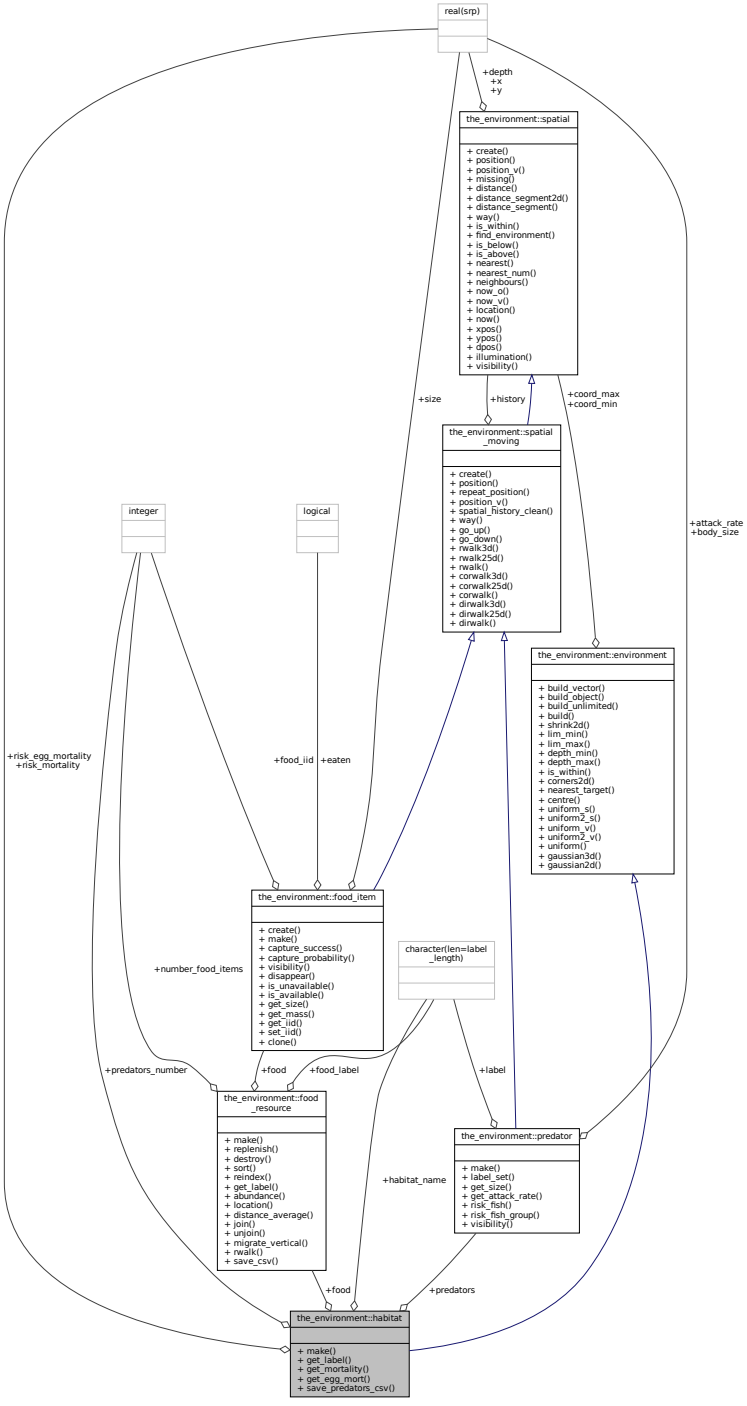
9.32 the_environment::habitat Type Reference

Definition of the **environment habitat** `HABITAT` object. There can potentially be of several types of habitats (patches etc.), so the superclass HABITAT defines the most general properties and procedures. More specific procedures are defined in sub-objects. Such procedures can be overriden from super-object to sub-objects providing for procedure polymorphism.

Inheritance diagram for the_environment::habitat:



Collaboration diagram for the_environment::habitat:



Public Member Functions

- procedure, public make => habitat_make_init

  Make an instance of the habitat object. See `the_environment::habitat_make_init()`
- procedure, public get_label => habitat_name_get

  Return the name (label) of the habitat. See `the_environment::habitat_name_get()`.
- procedure, public get_mortality => habitat_get_risk_mortality

  Get the mortality risk associated with this habitat. See `the_environment::habitat_get_risk_mortality()`.

- procedure, public get_egg_mort => habitat_get_risk_mortality_egg

  Get the egg mortality risk associated with this habitat. See `the_environment::habitat_get_risk_mortality_egg()`.
- procedure, public save_predators_csv => habitat_save_predators_csv

  Save the predators with their characteristics into a CSV file. See `the_environment::habitat_save_predators_csv()`.

Public Attributes

- character(len=label_length) habitat_name

  The name of the habitat.
- real(srp) risk_mortality

  Other agent mortality risks.
- real(srp) risk_egg_mortality

  Egg mortality risk.
- integer predators_number

  Number of predators that dwell in the habitat.
- type(predator), dimension(:), allocatable predators

  Habitat has an array of predators (i.e. `PREDATOR` *objects).*
- type(food_resource) food

  Habitat has a food resource (i.e. FOOD_RESOURCE`object) which is an array of`*FOOD_ITEM*`s.` `@note A container object`*FOOD_RESOURCE`is used for the food resource rather than just raw number of food items and array of food items (as done with predation) to allow implementation of several different food resources more easily.*

9.32.1 Detailed Description

Definition of the **environment habitat** `HABITAT` object. There can potentially be of several types of habitats (patches etc.), so the superclass HABITAT defines the most general properties and procedures. More specific procedures are defined in sub-objects. Such procedures can be overriden from super-object to sub-objects providing for procedure polymorphism.
Definition at line 555 of file m_env.f90.

9.32.2 Member Function/Subroutine Documentation

9.32.2.1 make()

`procedure, public the_environment::habitat::make`
Make an instance of the habitat object. See `the_environment::habitat_make_init()`
Definition at line 587 of file m_env.f90.

9.32.2.2 get_label()

`procedure, public the_environment::habitat::get_label`
Return the name (label) of the habitat. See `the_environment::habitat_name_get()`.
Definition at line 590 of file m_env.f90.

9.32.2.3 get_mortality()

`procedure, public the_environment::habitat::get_mortality`
Get the mortality risk associated with this habitat. See `the_environment::habitat_get_risk_mortality()`.
Definition at line 593 of file m_env.f90.

9.32.2.4 get_egg_mort()

procedure, public the_environment::habitat::get_egg_mort
Get the egg mortality risk associated with this habitat. See the_environment::habitat_get_risk_mortality_egg().
Definition at line 596 of file m_env.f90.

9.32.2.5 save_predators_csv()

procedure, public the_environment::habitat::save_predators_csv
Save the predators with their characteristics into a CSV file. See the_environment::habitat_save_predators_csv().
Definition at line 599 of file m_env.f90.

9.32.3 Member Data Documentation

9.32.3.1 habitat_name

character (len=label_length) the_environment::habitat::habitat_name
The name of the habitat.
Definition at line 557 of file m_env.f90.

9.32.3.2 risk_mortality

real(srp) the_environment::habitat::risk_mortality
Other agent mortality risks.
Definition at line 559 of file m_env.f90.

9.32.3.3 risk_egg_mortality

real(srp) the_environment::habitat::risk_egg_mortality
Egg mortality risk.
Definition at line 561 of file m_env.f90.

9.32.3.4 predators_number

integer the_environment::habitat::predators_number
Number of predators that dwell in the habitat.
Definition at line 564 of file m_env.f90.

9.32.3.5 predators

type(predator), dimension(:), allocatable the_environment::habitat::predators
Habitat has an array of predators (i.e. PREDATOR objects).

Note

The implementation of predators is very simplistic here, just a single type of predators integrated into the HABITAT object, without a separate predator container. This is, for example, different from the food resources made as a FOOD_RESOURCE container (below) that allows several types of food. A more advanced version should implement a specific container like FOOD_RESOURCE and, ultimately, a full implementation of an AHA predator (with the genome, neurobiology etc.). Do we need several types of predators or predation bound functions?

Definition at line 575 of file m_env.f90.

9.32.3.6 food

type(food_resource) the_environment::habitat::food
Habitat has a food resource (i.e. FOOD_RESOURCEobject) which is an array ofFOOD_ITEMs. @note A container objectFOOD_RESOURCE`is used for the food resource rather than just raw number of food items and array of food items (as done with predation) to allow implementation of several different food resources more easily.
Definition at line 583 of file m_env.f90.
The documentation for this type was generated from the following file:

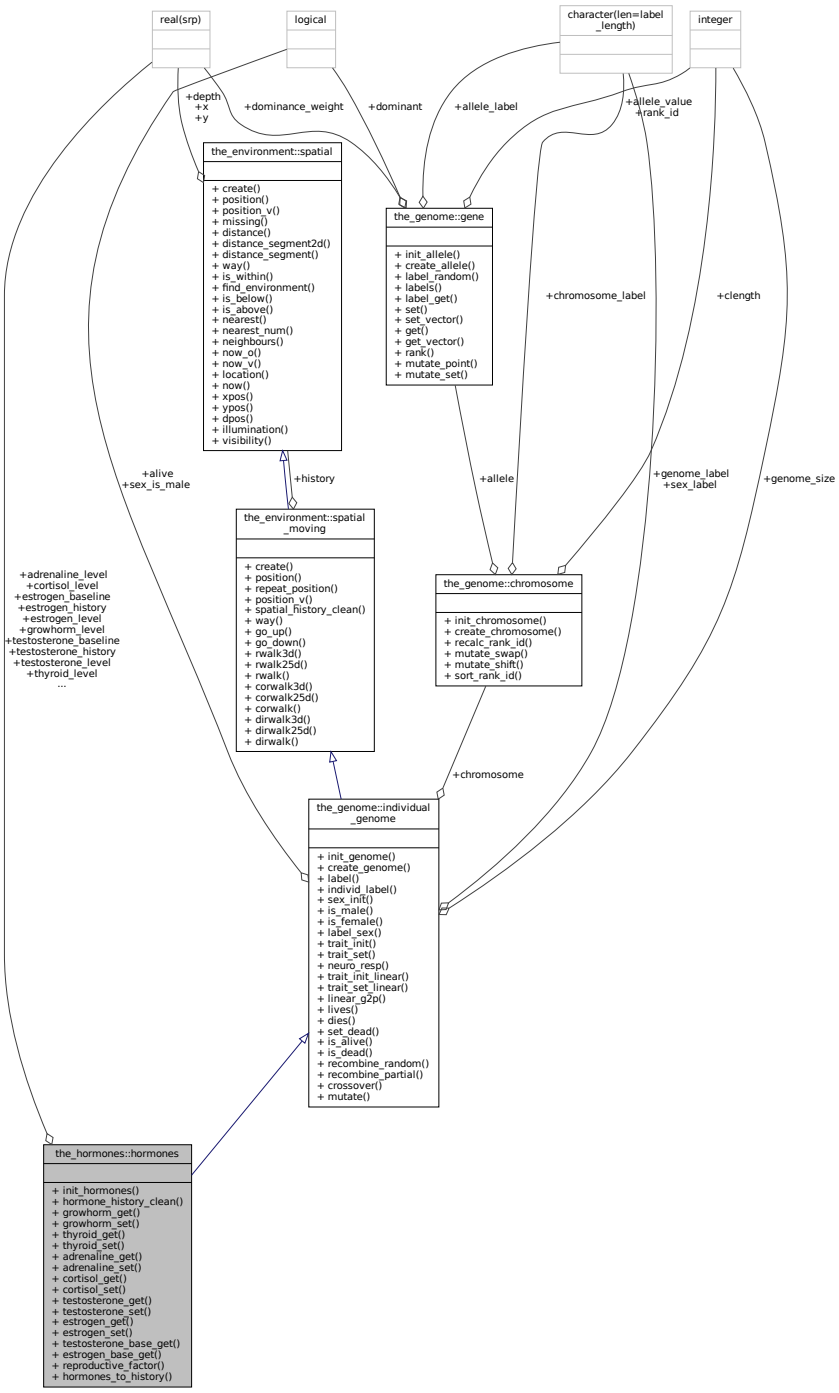
- m_env.f90

9.33 the_hormones::hormones Type Reference

This type adds hormonal architecture extending the genome object.

Inheritance diagram for the_hormones::hormones:

Collaboration diagram for the_hormones::hormones:



Public Member Functions

- procedure, public init_hormones => hormones_init_genotype

  Initialise hormone levels based on the genome value. See `the_hormones::hormones_init_genotype()`
- procedure, public hormone_history_clean => hormones_clean_history_stack

  Clean the history stack of hormones. See `the_hormones::hormones_clean_history_stack()`
- procedure, public growhorm_get => growhorm_get_level

  Get the value of ***thyroid****. See* `the_hormones::growhorm_get_level()`

- procedure, public growhorm_set => growhorm_set_level

  *Set the value of **thyroid**. See `the_hormones::growhorm_set_level()`*
- procedure, public thyroid_get => thyroid_get_level

  *Get the value of **thyroid**. See `the_hormones::thyroid_get_level()`*
- procedure, public thyroid_set => thyroid_set_level

  *Set the value of **thyroid**. See `the_hormones::thyroid_set_level()`*
- procedure, public adrenaline_get => adrenaline_get_level

  *Get the value of **adrenaline**. See `the_hormones::adrenaline_get_level()`*
- procedure, public adrenaline_set => adrenaline_set_level

  *Set the value of **adrenaline**. See `the_hormones::adrenaline_set_level()`*
- procedure, public cortisol_get => cortisol_get_level

  *Get the value of **cortisol**. See `the_hormones::cortisol_get_level()`*
- procedure, public cortisol_set => cortisol_set_level

  *Set the value of **cortisol**. See `the_hormones::cortisol_set_level()`*
- procedure, public testosterone_get => testosterone_get_level

  *Get the value of **testosterone**. See `the_hormones::testosterone_get_level()`*
- procedure, public testosterone_set => testosterone_set_level

  *Set the value of **testosterone**. See `the_hormones::testosterone_set_level()`*
- procedure, public estrogen_get => estrogen_get_level

  *Get the value of **estrogen**. See `the_hormones::estrogen_get_level()`*
- procedure, public estrogen_set => estrogen_set_level

  *Set the value of **estrogen**. See `the_hormones::estrogen_set_level()`*
- procedure, public testosterone_base_get => testosteron_baseline_get_level

  Get the value of testosterone baseline. See `the_hormones::testosteron_baseline_get_level()`
- procedure, public estrogen_base_get => estrogen_baseline_get_level

  Get the value of estrogen baseline. See `the_hormones::estrogen_baseline_get_level()`
- procedure, public reproductive_factor => hormones_reproductive_factor_calc

  Calculate the reproductive factor. Reproductive factor is defined as the current level of the_hormones::testosterone↩ _level in males and the_hormones::estrogen_level in females. See `the_hormones::hormones_reproductive_factor_cal`
- procedure, public hormones_to_history => hormones_update_history

  Update the sex steroid hormones history stack from the current level See `the_hormones::hormones_update_history()`.

Public Attributes

- real(srp) growhorm_level

  ***growth** hormone increases metabolic rate and growth, has costs changes/effects relatively slow and long-term.*
- real(srp) thyroid_level

  ***thyroid** hormone limits growth hormone, has costs changes/effects very slow, level very stable, genetically determined.*
- real(srp) adrenaline_level

  ***adrenaline** increases general arousal, increases escape speed/performance, primes active fear response, primes aggression, increases cognitive performance, focus attention, suppresses immune system, changes/effects relatively short-term.*
- real(srp) cortisol_level

  ***cortisol** (HPI axis) linked with higher stress and fear, reduces hunger, suppresses immune system, increases blood pressure, reduce cognitive performance, changes/effects long-term.*
- real(srp) testosterone_level

  Gonadal steroids - Sex hormones of males and females.
- real(srp) estrogen_level

  ***estrogen** - development of female sex characteristics, suppresses immunity, changes/effects relatively short-term.*
- real(srp) testosterone_baseline

  The testosterone baseline genetically determined level.

- real(srp) estrogen_baseline

  The estrogen baseline genetically determined level.
- real(srp), dimension(history_size_agent_prop) testosterone_history

  History stacks for the gonadal steroids.
- real(srp), dimension(history_size_agent_prop) estrogen_history

9.33.1 Detailed Description

This type adds hormonal architecture extending the genome object.
Definition at line 28 of file m_hormon.f90.

9.33.2 Member Function/Subroutine Documentation

9.33.2.1 init_hormones()

`procedure, public the_hormones::hormones::init_hormones`
Initialise hormone levels based on the genome value. See `the_hormones::hormones_init_genotype()`
Definition at line 70 of file m_hormon.f90.

9.33.2.2 hormone_history_clean()

`procedure, public the_hormones::hormones::hormone_history_clean`
Clean the history stack of hormones. See `the_hormones::hormones_clean_history_stack()`
Definition at line 73 of file m_hormon.f90.

9.33.2.3 growhorm_get()

`procedure, public the_hormones::hormones::growhorm_get`
Get the value of **thyroid**. See `the_hormones::growhorm_get_level()`
Definition at line 77 of file m_hormon.f90.

9.33.2.4 growhorm_set()

`procedure, public the_hormones::hormones::growhorm_set`
Set the value of **thyroid**. See `the_hormones::growhorm_set_level()`
Definition at line 80 of file m_hormon.f90.

9.33.2.5 thyroid_get()

`procedure, public the_hormones::hormones::thyroid_get`
Get the value of **thyroid**. See `the_hormones::thyroid_get_level()`
Definition at line 84 of file m_hormon.f90.

9.33.2.6 thyroid_set()

`procedure, public the_hormones::hormones::thyroid_set`
Set the value of **thyroid**. See `the_hormones::thyroid_set_level()`
Definition at line 87 of file m_hormon.f90.

9.33.2.7 adrenaline_get()

procedure, public the_hormones::hormones::adrenaline_get
Get the value of **adrenaline**. See the_hormones::adrenaline_get_level()
Definition at line 91 of file m_hormon.f90.

9.33.2.8 adrenaline_set()

procedure, public the_hormones::hormones::adrenaline_set
Set the value of **adrenaline**. See the_hormones::adrenaline_set_level()
Definition at line 94 of file m_hormon.f90.

9.33.2.9 cortisol_get()

procedure, public the_hormones::hormones::cortisol_get
Get the value of **cortisol**. See the_hormones::cortisol_get_level()
Definition at line 98 of file m_hormon.f90.

9.33.2.10 cortisol_set()

procedure, public the_hormones::hormones::cortisol_set
Set the value of **cortisol**. See the_hormones::cortisol_set_level()
Definition at line 101 of file m_hormon.f90.

9.33.2.11 testosterone_get()

procedure, public the_hormones::hormones::testosterone_get
Get the value of **testosterone**. See the_hormones::testosterone_get_level()
Definition at line 105 of file m_hormon.f90.

9.33.2.12 testosterone_set()

procedure, public the_hormones::hormones::testosterone_set
Set the value of **testosterone**. See the_hormones::testosterone_set_level()
Definition at line 108 of file m_hormon.f90.

9.33.2.13 estrogen_get()

procedure, public the_hormones::hormones::estrogen_get
Get the value of **estrogen**. See the_hormones::estrogen_get_level()
Definition at line 112 of file m_hormon.f90.

9.33.2.14 estrogen_set()

procedure, public the_hormones::hormones::estrogen_set
Set the value of **estrogen**. See the_hormones::estrogen_set_level()
Definition at line 115 of file m_hormon.f90.

9.33.2.15 testosterone_base_get()

procedure, public the_hormones::hormones::testosterone_base_get
Get the value of testosterone baseline. See the_hormones::testosteron_baseline_get_level()

Definition at line 119 of file m_hormon.f90.

9.33.2.16 estrogen_base_get()

`procedure, public the_hormones::hormones::estrogen_base_get`
Get the value of estrogen baseline. See `the_hormones::estrogen_baseline_get_level()`
Definition at line 123 of file m_hormon.f90.

9.33.2.17 reproductive_factor()

`procedure, public the_hormones::hormones::reproductive_factor`
Calculate the reproductive factor. Reproductive factor is defined as the current level of the_hormones↩ ::testosterone_level in males and the_hormones::estrogen_level in females. See `the_hormones::hormones_reproductive`
Definition at line 129 of file m_hormon.f90.

9.33.2.18 hormones_to_history()

`procedure, public the_hormones::hormones::hormones_to_history`
Update the sex steroid hormones history stack from the current level See `the_hormones::hormones_update_history()`.
Definition at line 134 of file m_hormon.f90.

9.33.3 Member Data Documentation

9.33.3.1 growhorm_level

`real(srp) the_hormones::hormones::growhorm_level`
growth hormone increases metabolic rate and growth, has costs changes/effects relatively slow and long-term.
Definition at line 32 of file m_hormon.f90.

9.33.3.2 thyroid_level

`real(srp) the_hormones::hormones::thyroid_level`
thyroid hormone limits growth hormone, has costs changes/effects very slow, level very stable, genetically determined.
Definition at line 35 of file m_hormon.f90.

9.33.3.3 adrenaline_level

`real(srp) the_hormones::hormones::adrenaline_level`
adrenaline increases general arousal, increases escape speed/performance, primes active fear response, primes aggression, increases cognitive performance, focus attention, suppresses immune system, changes/effects relatively short-term.
Definition at line 40 of file m_hormon.f90.

9.33.3.4 cortisol_level

`real(srp) the_hormones::hormones::cortisol_level`
cortisol (HPI axis) linked with higher stress and fear, reduces hunger, suppresses immune system, increases blood pressure, reduce cognitive performance, changes/effects long-term.
Definition at line 44 of file m_hormon.f90.

9.33.3.5 testosterone_level

`real(srp) the_hormones::hormones::testosterone_level`
Gonadal steroids - Sex hormones of males and females.

- we implement them and their effects differently i males and females **testosterone** - development of male sex characteristics increases boldness and aggression, reduces immunity, changes/effects relatively short-term

  Note

  use *testosterone* (with *e*), not *testosteron* in the code!
  single-hormone initialisation function is private, we don't need to init single hormones anywhere outside of this module.

Definition at line 54 of file m_hormon.f90.

9.33.3.6 estrogen_level

`real(srp) the_hormones::hormones::estrogen_level`
estrogen - development of female sex characteristics, suppresses immunity, changes/effects relatively short-term.
Definition at line 57 of file m_hormon.f90.

9.33.3.7 testosterone_baseline

`real(srp) the_hormones::hormones::testosterone_baseline`
The *testosterone* baseline genetically determined level.
Definition at line 59 of file m_hormon.f90.

9.33.3.8 estrogen_baseline

`real(srp) the_hormones::hormones::estrogen_baseline`
The *estrogen* baseline genetically determined level.
Definition at line 61 of file m_hormon.f90.

9.33.3.9 testosterone_history

`real(srp), dimension(history_size_agent_prop) the_hormones::hormones::testosterone_history`
History stacks for the gonadal steroids.
Definition at line 63 of file m_hormon.f90.

9.33.3.10 estrogen_history

`real(srp), dimension(history_size_agent_prop) the_hormones::hormones::estrogen_history`
Definition at line 64 of file m_hormon.f90.
The documentation for this type was generated from the following file:

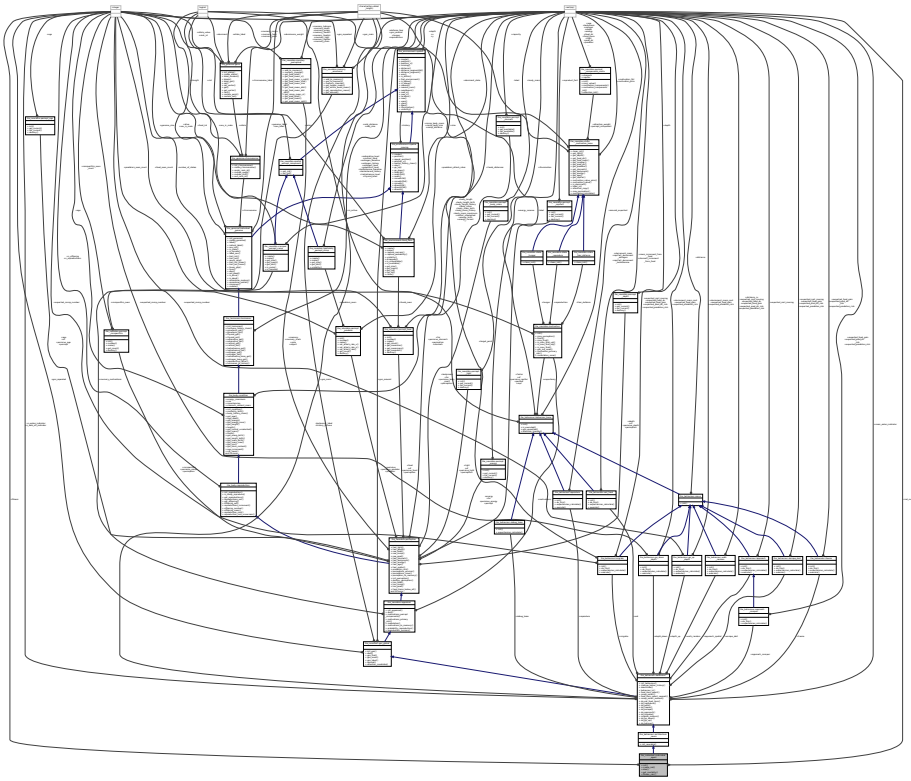
- m_hormon.f90

9.34 the_individual::individual_agent Type Reference

This type describes parameters of the individual agent.

Inheritance diagram for the_individual::individual_agent:

Collaboration diagram for the_individual::individual_agent:



Public Member Functions

- procedure, public init => individual_init_random

  Generate a random agent from the genotype. See `the_individual::individual_init_random()`.
- procedure, public create_ind => individual_create_zero

  Generate a new empty agent. See `the_individual::individual_create_zero()`.
- procedure, public dies => individual_agent_set_dead

  *Set the individual to be **dead**. This method overrides the the_genome::individual_genome::dies() method, nullifying all reproductive and neurobiological and behavioural objects. However, this function does not deallocate the individual agent object, this may be a separate destructor function. The* `dies` *method is implemented at the following levels of the agent object hierarchy (upper overrides the lower level):*
- procedure, public get_mortality => individual_get_risk_mortality_individual

  Get the individually-specific mortality risk for the agent. See `the_individual::individual_get_risk_mortality_indiv`
- procedure, public fitness_calc => individual_preevol_fitness_calc

  Calculate fitness for the pre-evolution phase of the genetic algorithm. Pre-evolution is based on selection for a simple criterion without explicit reproduction etc. The criterion for selection at this phase is set by the integer the_individual::individual_agent::fitness component. See `the_individual::individual_preevol_fitness_calc()`.

Public Attributes

- integer fitness

  fitness is a fixed criterion for the evolution at the starting 'pre-evolution' phase of the genetic algorithm.
- real(srp) ind_mortality

  Individually specific mortality risk for various undefined or defined reasons, such as immune status. Note that this risk is not linked to the predation or habitat-specific factors. The latter two causes are treated separately.

9.34.1 Detailed Description

This type describes parameters of the individual agent.
`INDIVIDUAL_AGENT` extends the neurobiological architecture type by adding the alive flag and explicit fitness. Fitness can be either set or calculated or assessed approximately. In models, it can be used in calculations or used just for data output. NOTE: procedure, final may not be implemented in all compilers. E.g. gfortran issues this error prior to 4.9: Error: Finalization at (1) is not yet implemented

Note

`INDIVIDUAL_AGENT` is an umbrella object for the individual

Definition at line 37 of file m_indiv.f90.

9.34.2 Member Function/Subroutine Documentation

9.34.2.1 init()

`procedure, public the_individual::individual_agent::init`
Generate a random agent from the genotype. See `the_individual::individual_init_random()`.
Definition at line 52 of file m_indiv.f90.

9.34.2.2 create_ind()

`procedure, public the_individual::individual_agent::create_ind`
Generate a new *empty* agent. See `the_individual::individual_create_zero()`.
Definition at line 55 of file m_indiv.f90.

9.34.2.3 dies()

`procedure, public the_individual::individual_agent::dies`
Set the individual to be **dead**. This method overrides the the_genome::individual_genome::dies() method, nullifying all reproductive and neurobiological and behavioural objects. However, this function does not deallocate the individual agent object, this may be a separate destructor function. The `dies` method is implemented at the following levels of the agent object hierarchy (upper overrides the lower level):

- the_genome::individual_genome::dies();
- the_neurobio::appraisal::dies();
- the_neurobio::gos_global::dies();
- the_individual::individual_agent::dies().

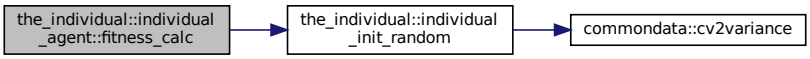
See `the_individual::individual_agent_set_dead()`.
Definition at line 69 of file m_indiv.f90.

9.34.2.4 get_mortality()

`procedure, public the_individual::individual_agent::get_mortality`
Get the individually-specific mortality risk for the agent. See `the_individual::individual_get_risk_mortality_inc`
Definition at line 72 of file m_indiv.f90.

9.34.2.5 fitness_calc()

`procedure, public the_individual::individual_agent::fitness_calc`
Calculate fitness for the pre-evolution phase of the genetic algorithm. Pre-evolution is based on selection for a simple criterion without explicit reproduction etc. The criterion for selection at this phase is set by the integer the_individual::individual_agent::fitness component. See `the_individual::individual_preevol_fitness_calc()`.
Definition at line 79 of file m_indiv.f90.
Here is the call graph for this function:



9.34.3 Member Data Documentation

9.34.3.1 fitness

`integer the_individual::individual_agent::fitness`
fitness is a fixed criterion for the evolution at the starting 'pre-evolution' phase of the genetic algorithm.

Warning

Note that fitness here is actually an "antifitness", the higher its value, the **worse** fitting is the agent.

Definition at line 42 of file m_indiv.f90.

9.34.3.2 ind_mortality

`real(srp) the_individual::individual_agent::ind_mortality`
Individually specific mortality risk for various undefined or defined reasons, such as immune status. Note that this risk is not linked to the predation or habitat-specific factors. The latter two causes are treated separately.
Definition at line 47 of file m_indiv.f90.
The documentation for this type was generated from the following file:

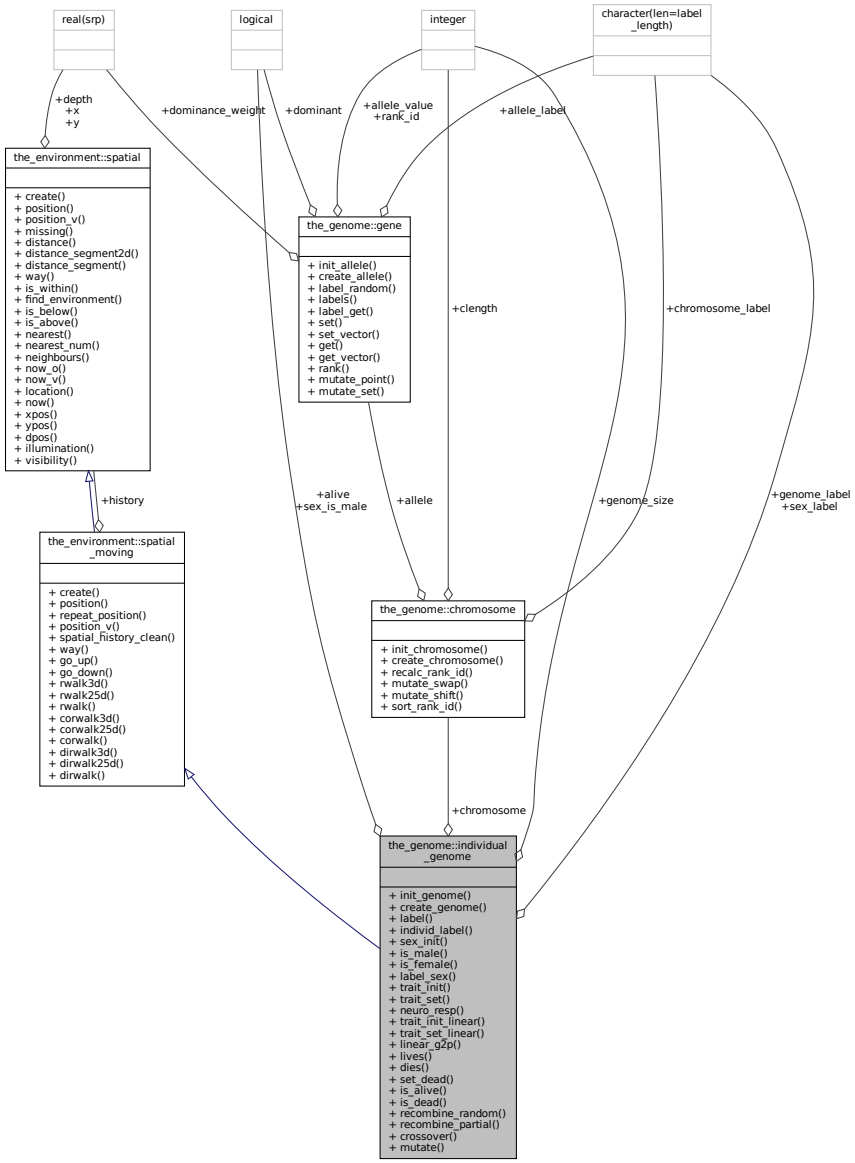
- m_indiv.f90

9.35 the_genome::individual_genome Type Reference

This type describes parameters of the individual agent's genome The genome is an array of allocatable the_genome::chromosome objects, different kinds of agents may have different genomes with different number of chromosomes. See "the genome structure" for as general description and "genome" for details.

Inheritance diagram for the_genome::individual_genome:

Collaboration diagram for the_genome::individual_genome:



Public Member Functions

- procedure, public init_genome => genome_init_random

  Initialise the genome at random, and set sex as determined by the sex determination locus. See `the_genome::genome_init_random()`
- procedure, public create_genome => genome_create_zero

  Create a new empty genome, and set sex as determined by the sex determination locus. Genome values are from parents using inherit functions. See `the_genome::genome_create_zero()`
- procedure, public label => genome_label_set

  Label genome. If label is not provided, make a random string.
- procedure, public individ_label => genome_label_get

  Accessor function to get the genome label. The label is a kind of a (random) text string name of the genome and the individual agent.
- procedure, public sex_init => genome_sex_determine_init

Sex has a separate status from all other genetically determined traits. It is initialised here, at the genotype level of the class hierarchy. See `the_genome::genome_sex_determine_init()`

- procedure, public is_male => genome_get_sex_is_male

  Get the logical sex ID of the genome object component. See `the_genome::genome_get_sex_is_male()`
- procedure, public is_female => genome_get_sex_is_female

  Get the logical sex ID of the genome object component. See `the_genome::genome_get_sex_is_female()`
- procedure, public label_sex => genome_get_sex_label

  Get the descriptive sex label: male or female. See `the_genome::genome_get_sex_label()`
- procedure, public trait_init => trait_init_genotype_gamma2gene

  Init a trait from the genotype, trait can be any object in any of the up level class hierarchy that is determined from the boolean genotype x phenotype matrix. See `the_genome::trait_init_genotype_gamma2gene()`
- procedure, public trait_set => trait_set_genotype_gamma2gene

  *Set an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy. See* `the_genome::trait_set_genotype_gamma2gene()`
- generic, public neuro_resp => trait_init, trait_set

  Generic interface to the neuronal response function. See `the_genome::trait_init_genotype_gamma2gene()` *and* `the_genome::trait_set_genotype_gamma2gene()`.
- procedure, public trait_init_linear => trait_init_linear_sum_additive_comps_2genes_r

  Init a trait from the genotype, trait can be any object in any of the up level class hierarchy that is determined from the boolean genotype x phenotype matrix. Note that this method is based on simple linear rescale rather than neuronal response. See `the_genome::trait_init_linear_sum_additive_comps_2genes_r()`
- procedure, public trait_set_linear => trait_set_linear_sum_additive_comps_2genes_r

  *Set an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy. Note that this method is based on simple linear rescale rather than neuronal response. See* `the_genome::trait_set_linear_sum_additive_comps_2genes_r()`
- generic, public linear_g2p => trait_init_linear, trait_set_linear

  Generic interface to the simple linear genotype to phenotype transformation functions. See `the_genome::trait_init_linear_s` *and* `the_genome::trait_set_linear_sum_additive_comps_2genes_r()`.
- procedure, public lives => genome_individual_set_alive

  Set the individual to be alive, normally this function is used after init or birth. See `the_genome::genome_individual_set_aliv`
- procedure, public dies => genome_individual_set_dead

  *Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function. The* `dies` *method is implemented at the following levels of the agent object hierarchy (upper overrides the lower level):*
- procedure, public set_dead => genome_individual_set_dead

  *Set the individual to be **dead**. Note that in this class this method implementation points to the same procedure as* `the_genome::individual_genome::dies()`. *However the*dies` *method is overriden upwards in the class hierarchy to also nullify neurobiological and behavioural objects. So this method should be only called in procedures that specifically implemented override of the* `dies` *method:*
- procedure, public is_alive => genome_individual_check_alive

  Check if the individual is alive. See `the_genome::genome_individual_check_alive()`
- procedure, public is_dead => genome_individual_check_dead

  Check if the individual is dead (the opposite of `is_alive`*). See* `the_genome::genome_individual_check_dead()`
- procedure, public recombine_random => genome_individual_recombine_homol_full_rand_alleles

  *Internal **genetic recombination backend**, exchange individual alleles between homologous chromosomes in mother and father genomes to form the* `this` *(offspring) genome. Fully random recombination. See* `the_genome::genome_individual_recombine_homol_full_rand_alleles()`.
- procedure, public recombine_partial => genome_individual_recombine_homol_part_rand_alleles

  Internal genetic recombination backend, exchange individual alleles between homologous chromosomes in mother and father genomes to form the `this` *(offspring) genome. Partially random recombination. See* `the_genome::genome_individual_recombine_homol_part_rand_alleles()`.
- procedure, public crossover => genome_individual_crossover_homol_fix

  *Internal **fixed genetic crossover** backend, exchange blocks of alleles between homologous chromosomes in mother and father genomes to form the* `this` *(offspring) genome. See* `the_genome::genome_individual_crossover_homol_fix`
- procedure, public mutate => genome_mutate_wrapper

  Perform a probabilistic random mutation(s) on the individual genome. This is a high level wrapper to build mutations from various components. See `the_genome::genome_mutate_wrapper()`.

Public Attributes

- character(len=label_length) genome_label

  label for the genome
- integer genome_size = N_CHROMOSOMES

  the size of the genome, i.e. N of chromosomes = N_CHROMOSOMES in this version it is constant, can implement variable genomes later.
- type(chromosome), dimension(:,:), allocatable chromosome

  array of chromosome objects, the two dimensions refer to (1) chromosome number in the genome and (2) the number of homologs (1:2 for diploid), so chromosome is a 2D array.
- logical sex_is_male

  The sex of the individual: is male = TRUE or female = FALSE this is the main sex identifier. The sex_label defined below should only be used for outputs and similar purposes.
- character(len=label_length) sex_label

  Verbal label for sex ("male" or "female").
- logical alive

  Flag the agent is alive (TRUE) or dead (False).

9.35.1 Detailed Description

This type describes parameters of the individual agent's genome The genome is an array of allocatable the_genome::chromosome objects, different kinds of agents may have different genomes with different number of chromosomes. See "the genome structure" for as general description and "genome" for details.
Definition at line 160 of file m_genome.f90.

9.35.2 Member Function/Subroutine Documentation

9.35.2.1 init_genome()

`procedure, public the_genome::individual_genome::init_genome`
Initialise the genome at random, and set sex as determined by the sex determination locus. See `the_genome::genome_init_random()`
Definition at line 182 of file m_genome.f90.

9.35.2.2 create_genome()

`procedure, public the_genome::individual_genome::create_genome`
Create a new empty genome, and set sex as determined by the sex determination locus. Genome values are from parents using inherit functions. See `the_genome::genome_create_zero()`
Definition at line 187 of file m_genome.f90.

9.35.2.3 label()

`procedure, public the_genome::individual_genome::label`
Label genome. If label is not provided, make a random string.

Note

TMP NOTE: label setting removed from the create function as it will now override create for `SPATIAL_MOVING`. See `the_genome::genome_label_set()`

Definition at line 192 of file m_genome.f90.

9.35.2.4 individ_label()

procedure, public the_genome::individual_genome::individ_label
Accessor function to get the genome label. The label is a kind of a (random) text string name of the genome and the individual agent.

Note

We especially need this accessor function because the genome (and individual) name is used in other modules for file names ids etc. See the_genome::genome_label_get()

Definition at line 198 of file m_genome.f90.

9.35.2.5 sex_init()

procedure, public the_genome::individual_genome::sex_init
Sex has a separate status from all other genetically determined traits. It is initialised here, at the genotype level of the class hierarchy. See the_genome::genome_sex_determine_init()
Definition at line 203 of file m_genome.f90.

9.35.2.6 is_male()

procedure, public the_genome::individual_genome::is_male
Get the logical sex ID of the genome object component. See the_genome::genome_get_sex_is_male()
Definition at line 206 of file m_genome.f90.

9.35.2.7 is_female()

procedure, public the_genome::individual_genome::is_female
Get the logical sex ID of the genome object component. See the_genome::genome_get_sex_is_female()
Definition at line 209 of file m_genome.f90.

9.35.2.8 label_sex()

procedure, public the_genome::individual_genome::label_sex
Get the descriptive sex label: male or female. See the_genome::genome_get_sex_label()
Definition at line 212 of file m_genome.f90.

9.35.2.9 trait_init()

procedure, public the_genome::individual_genome::trait_init
Init a trait from the genotype, trait can be any object in any of the up level class hierarchy that is determined from the boolean genotype x phenotype matrix. See the_genome::trait_init_genotype_gamma2gene()
Definition at line 218 of file m_genome.f90.

9.35.2.10 trait_set()

procedure, public the_genome::individual_genome::trait_set
Set an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy. See the_genome::trait_set_genotype_gamma2gene()
Definition at line 222 of file m_genome.f90.

9.35.2.11 neuro_resp()

generic, public the_genome::individual_genome::neuro_resp
Generic interface to the neuronal response function. See the_genome::trait_init_genotype_gamma2gene() and the_genome::trait_set_genotype_gamma2gene().
Definition at line 226 of file m_genome.f90.

9.35.2.12 trait_init_linear()

procedure, public the_genome::individual_genome::trait_init_linear
Init a trait from the genotype, trait can be any object in any of the up level class hierarchy that is determined from the boolean genotype x phenotype matrix. Note that this method is based on simple linear rescale rather than neuronal response. See the_genome::trait_init_linear_sum_additive_comps_2genes_r()
Definition at line 233 of file m_genome.f90.

9.35.2.13 trait_set_linear()

procedure, public the_genome::individual_genome::trait_set_linear
Set an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy. Note that this method is based on simple linear rescale rather than neuronal response. See the_genome::trait_set_linear_sum_additive_comps_2genes_r()
Definition at line 240 of file m_genome.f90.

9.35.2.14 linear_g2p()

generic, public the_genome::individual_genome::linear_g2p
Generic interface to the simple linear genotype to phenotype transformation functions. See the_genome::trait_init_linear and the_genome::trait_set_linear_sum_additive_comps_2genes_r().
Definition at line 246 of file m_genome.f90.

9.35.2.15 lives()

procedure, public the_genome::individual_genome::lives
Set the individual to be alive, normally this function is used after init or birth. See the_genome::genome_individual_set_al
Definition at line 251 of file m_genome.f90.

9.35.2.16 dies()

procedure, public the_genome::individual_genome::dies
Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function. The dies method is implemented at the following levels of the agent object hierarchy (upper overrides the lower level):

- the_genome::individual_genome::dies();
- the_neurobio::appraisal::dies();
- the_neurobio::gos_global::dies();
- the_individual::individual_agent::dies().

See the_genome::genome_individual_set_dead()
Definition at line 263 of file m_genome.f90.

9.35.2.17 set_dead()

procedure, public the_genome::individual_genome::set_dead
Set the individual to be **dead**. Note that in this class this method implementation points to the same procedure as the_genome::individual_genome::dies(). However thedies` method is overriden upwards in the class hierarchy to also nullify neurobiological and behavioural objects. So this method should be only called in procedures that specifically implemented override of the dies method:

- the_neurobio::gos_global::dies()
- the_individual::individual_agent::dies()

See the_genome::genome_individual_set_dead()
Definition at line 275 of file m_genome.f90.

9.35.2.18 is_alive()

procedure, public the_genome::individual_genome::is_alive
Check if the individual is alive. See the_genome::genome_individual_check_alive()
Definition at line 278 of file m_genome.f90.

9.35.2.19 is_dead()

procedure, public the_genome::individual_genome::is_dead
Check if the individual is dead (the opposite of is_alive). See the_genome::genome_individual_check_dead()
Definition at line 281 of file m_genome.f90.

9.35.2.20 recombine_random()

procedure, public the_genome::individual_genome::recombine_random
Internal **genetic recombination backend**, exchange individual alleles between homologous chromosomes in mother and father genomes to form the this (offspring) genome. Fully random recombination. See the_genome::genome_individual_recombine_homol_full_rand_alleles().
Definition at line 287 of file m_genome.f90.

9.35.2.21 recombine_partial()

procedure, public the_genome::individual_genome::recombine_partial
Internal genetic recombination backend, exchange individual alleles between homologous chromosomes in mother and father genomes to form the this (offspring) genome. Partially random recombination. See the_genome::genome_individual_recombine_homol_part_rand_alleles().
Definition at line 293 of file m_genome.f90.

9.35.2.22 crossover()

procedure, public the_genome::individual_genome::crossover
Internal **fixed genetic crossover** backend, exchange blocks of alleles between homologous chromosomes in mother and father genomes to form the this (offspring) genome. See the_genome::genome_individual_crossover_ho
Definition at line 299 of file m_genome.f90.

9.35.2.23 mutate()

procedure, public the_genome::individual_genome::mutate
Perform a probabilistic random mutation(s) on the individual genome. This is a high level wrapper to build mutations from various components. See the_genome::genome_mutate_wrapper().
Definition at line 305 of file m_genome.f90.

9.35.3 Member Data Documentation

9.35.3.1 genome_label

`character(len=label_length) the_genome::individual_genome::genome_label`
label for the genome
Definition at line 162 of file m_genome.f90.

9.35.3.2 genome_size

`integer the_genome::individual_genome::genome_size = N_CHROMOSOMES`
the size of the genome, i.e. N of chromosomes = N_CHROMOSOMES in this version it is constant, can implement variable genomes later.
Definition at line 165 of file m_genome.f90.

9.35.3.3 chromosome

`type(chromosome), dimension(:,:), allocatable the_genome::individual_genome::chromosome`
array of chromosome objects, the two dimensions refer to (1) chromosome number in the genome and (2) the number of homologs (1:2 for diploid), so chromosome is a 2D array.
Definition at line 169 of file m_genome.f90.

9.35.3.4 sex_is_male

`logical the_genome::individual_genome::sex_is_male`
The sex of the individual: is male = TRUE or female = FALSE this is the main sex identifier. The sex_label defined below should only be used for outputs and similar purposes.
Definition at line 173 of file m_genome.f90.

9.35.3.5 sex_label

`character(len=label_length) the_genome::individual_genome::sex_label`
Verbal label for sex ("male" or "female").
Definition at line 175 of file m_genome.f90.

9.35.3.6 alive

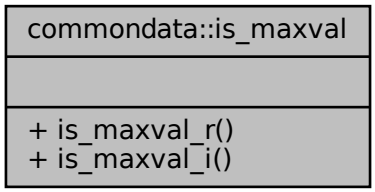
`logical the_genome::individual_genome::alive`
Flag the agent is alive (TRUE) or dead (False).
Definition at line 177 of file m_genome.f90.
The documentation for this type was generated from the following file:

- m_genome.f90

9.36 commondata::is_maxval Interface Reference

Check if a value is the maximum value of an array.

Collaboration diagram for commondata::is_maxval:



Public Member Functions

- pure logical function is_maxval_r (value, array, tolerance)

  Function to check if the value is the maximum value of an array (returns TRUE), or not (return FALSE).
- pure logical function is_maxval_i (value, array)

  Function to check if the value is the maximum value of an array (returns TRUE), or not (return FALSE). Integer version.

9.36.1 Detailed Description

Check if a value is the maximum value of an array.
Definition at line 5499 of file m_common.f90.

9.36.2 Member Function/Subroutine Documentation

9.36.2.1 is_maxval_r()

```
pure logical function commondata::is_maxval::is_maxval_r (
             real(srp), intent(in) value,
             real(srp), dimension(:), intent(in) array,
             real(srp), intent(in), optional tolerance )
```

Function to check if the value is the maximum value of an array (returns TRUE), or not (return FALSE).

Returns

TRUE if `value` is indeed the maximum value of the `array` and FALSE otherwise.

Parameters

`in`	*value*	The value to check
`in`	*array*	The array to check within.
`in`	*tolerance*	Optional tolerance threshold.

Definition at line 7459 of file m_common.f90.

9.36.2.2 is_maxval_i()

```
pure logical function commondata::is_maxval::is_maxval_i (
             integer, intent(in) value,
```

```
integer, dimension(:), intent(in) array )
```

Function to check if the value is the maximum value of an array (returns TRUE), or not (return FALSE). Integer version.

Returns

TRUE if `value` is indeed the maximum value of the `array` and FALSE otherwise.

Parameters

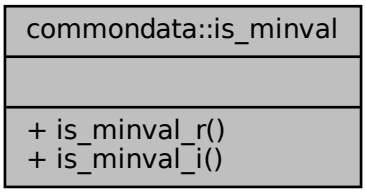
in	*value*	The value to check
in	*array*	The array to check within.

Definition at line 7497 of file m_common.f90.
The documentation for this interface was generated from the following file:

- m_common.f90

9.37 commondata::is_minval Interface Reference

Check if a value is the minimum value of an array.
Collaboration diagram for commondata::is_minval:



Public Member Functions

- pure logical function is_minval_r (value, array, tolerance)

  Function to check if the value is the minimum value of an array (returns TRUE), or not (return FALSE).
- pure logical function is_minval_i (value, array)

  Function to check if the value is the minimum value of an array (returns TRUE), or not (return FALSE). Integer version.

9.37.1 Detailed Description

Check if a value is the minimum value of an array.
Definition at line 5507 of file m_common.f90.

9.37.2 Member Function/Subroutine Documentation

9.37.2.1 is_minval_r()

```
pure logical function commondata::is_minval::is_minval_r (
            real(srp), intent(in) value,
```

```
real(srp), dimension(:), intent(in) array,
real(srp), intent(in), optional tolerance )
```

Function to check if the value is the minimum value of an array (returns TRUE), or not (return FALSE).

Returns

TRUE if `value` is indeed the minimum value of the `array` and FALSE otherwise.

Parameters

in	*value*	The value to check
in	*array*	The array to check within.
in	*tolerance*	Optional tolerance threshold.

Definition at line 7524 of file m_common.f90.

9.37.2.2 is_minval_i()

```
pure logical function commondata::is_minval::is_minval_i (
integer, intent(in) value,
integer, dimension(:), intent(in) array )
```

Function to check if the value is the minimum value of an array (returns TRUE), or not (return FALSE). Integer version.

Returns

TRUE if `value` is indeed the minimum value of the `array` and FALSE otherwise.

Parameters

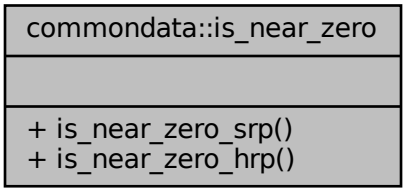
in	*value*	The value to check
in	*array*	The array to check within.

Definition at line 7562 of file m_common.f90.
The documentation for this interface was generated from the following file:

- m_common.f90

9.38 commondata::is_near_zero Interface Reference

Checks if a real number is near 0.0. Thus function can be used for comparing two real values like this:
Collaboration diagram for commondata::is_near_zero:



Public Member Functions

- elemental logical function is_near_zero_srp (test_number, epsilon)

  Checks if a real number is near 0.0. Thus function can be used for comparing two real values like the below.
- elemental logical function is_near_zero_hrp (test_number, epsilon)

  Checks if a real number is near 0.0. Thus function can be used for comparing two real values like the below.

9.38.1 Detailed Description

Checks if a real number is near 0.0. Thus function can be used for comparing two real values like this:
`if ( is_near_zero(a) ) then ...`
See the backend procedures commondata::is_near_zero_srp() and commondata::is_near_zero_hrp() for details.
Definition at line 5385 of file m_common.f90.

9.38.2 Member Function/Subroutine Documentation

9.38.2.1 is_near_zero_srp()

```
elemental logical function commondata::is_near_zero::is_near_zero_srp (
            real(srp), intent(in) test_number,
            real(srp), intent(in), optional epsilon )
```

Checks if a real number is near 0.0. Thus function can be used for comparing two real values like the below.
`if ( is_near_zero(a) ) then ...`

Note

Note that commondata::float_equal() can be used for approximate real comparisons of two real type values.

Modified from `Near0_dp()` function from Clerman & Spector 2012, p. 250-251.

This is the **standard** precision function (commondata::srp).

Parameters

in	*test_number*	test_number the number to check for being near-zero.
in	*epsilon*	epsilon optional (very small) tolerance value.

Returns

TRUE if the `test_number` is near-zero.

Definition at line 6202 of file m_common.f90.

9.38.2.2 is_near_zero_hrp()

```
elemental logical function commondata::is_near_zero::is_near_zero_hrp (
            real(hrp), intent(in) test_number,
            real(hrp), intent(in), optional epsilon )
```

Checks if a real number is near 0.0. Thus function can be used for comparing two real values like the below.
`if ( is_near_zero(a) ) then ...`

Note

Note that commondata::float_equal() can be used for approximate real comparisons of two real type values.

Modified from `Near0_dp()` function from Clerman & Spector 2012, p. 250-251.

This is the **high** precision function (commondata::hrp).

Parameters

in	*test_number*	test_number the number to check for being near-zero.
in	*epsilon*	epsilon optional (very small) tolerance value.

Returns

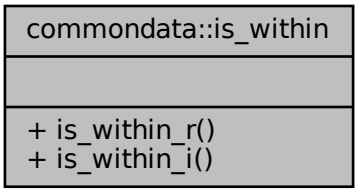
TRUE if the `test_number` is near-zero.

Definition at line 6232 of file m_common.f90.
The documentation for this interface was generated from the following file:

- m_common.f90

9.39 commondata::is_within Interface Reference

Logical function to check if a value is within a specific range, **lower <= X <= upper**.
Collaboration diagram for commondata::is_within:



Public Member Functions

- elemental logical function is_within_r (x, lower, upper)

  *Logical function to check if a value is within a specific range, **lower <= X <= upper**. The reverse (upper <= x <= lower) range limits can also be used; a corrective adjustment is automatically made.*
- elemental logical function is_within_i (x, lower, upper)

  *Logical function to check if a value is within a specific range, **lower <= X <= upper**. The reverse (upper <= x <= lower) range limits can also be used; a corrective adjustment is automatically made.*

9.39.1 Detailed Description

Logical function to check if a value is within a specific range, **lower <= X <= upper**.

Note

See commondata::is_within_operator_r() and commondata::is_within_operator_i() for a related user-defined operator `.within.`.

Definition at line 5362 of file m_common.f90.

9.39.2 Member Function/Subroutine Documentation

9.39.2.1 is_within_r()

```
elemental logical function commondata::is_within::is_within_r (
            real(srp), intent(in) x,
            real(srp), intent(in) lower,
            real(srp), intent(in) upper )
```

Logical function to check if a value is within a specific range, **lower <= X <= upper**. The reverse (upper <= x <= lower) range limits can also be used; a corrective adjustment is automatically made.

Note

This is the real type version.

See commondata::is_within_operator_r() and commondata::is_within_operator_i() for a related user-defined operator `.within.`.

Parameters

in	*x*	the value to test
in	*lower*	the lower limit for the range tested.
in	*upper*	the upper limit of the range tested.

Returns

Returns TRUE if `x` lies within [lower,upper] and FALSE otherwise.

Definition at line 5831 of file m_common.f90.

9.39.2.2 is_within_i()

```
elemental logical function commondata::is_within::is_within_i (
            integer, intent(in) x,
            integer, intent(in) lower,
            integer, intent(in) upper )
```

Logical function to check if a value is within a specific range, **lower <= X <= upper**. The reverse (upper <= x <= lower) range limits can also be used; a corrective adjustment is automatically made.

Note

This is the integer type version.

See commondata::is_within_operator_r() and commondata::is_within_operator_i() for a related user-defined operator `.within.`.

Parameters

in	*x*	the value to test
in	*lower*	the lower limit for the range tested.
in	*upper*	the upper limit of the range tested.

Returns

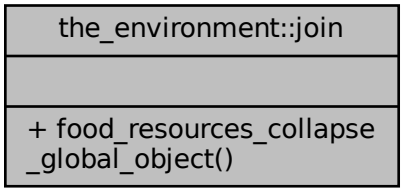
Returns TRUE if `x` lies within [lower,upper] and FALSE otherwise.

Definition at line 5868 of file m_common.f90.
The documentation for this interface was generated from the following file:

- m_common.f90

9.40 the_environment::join Interface Reference

An alias for the the_environment::food_resources_collapse_global_object() method for joining food resources into a single global food resource out of the global array the_environment::global_habitats_available. See the_environment::unjoin() for how to unjoin an array of food resources back into an array.
Collaboration diagram for the_environment::join:



Public Member Functions

- type(food_resource) function food_resources_collapse_global_object (reindex, label)

  Join food resources into a single global food resource out of the global array the_environment::global_habitats_available. See the_environment::unjoin() for how to unjoin an array of food resources back into an array. The joined resource can then go into the perception system. The properties of the component resources are retained in the collapsed resource.

9.40.1 Detailed Description

An alias for the the_environment::food_resources_collapse_global_object() method for joining food resources into a single global food resource out of the global array the_environment::global_habitats_available. See the_environment::unjoin() for how to unjoin an array of food resources back into an array.
Definition at line 767 of file m_env.f90.

9.40.2 Member Function/Subroutine Documentation

9.40.2.1 food_resources_collapse_global_object()

```
type(food_resource) function the_environment::join::food_resources_collapse_global_object (
            logical, intent(in), optional reindex,
            character(len=*), intent(in), optional label )
```

Join food resources into a single global food resource out of the global array the_environment::global_habitats_available. See the_environment::unjoin() for how to unjoin an array of food resources back into an array. The joined resource can then go into the perception system. The properties of the component resources are retained in the collapsed resource.

Note

This procedure is intended to use a short interface name `join`, see the_environment::join().

A similar procedure using a **list** of input component resources is implemented in the_environment::food_resources_collapse().

Parameters

in	*reindex*	reindex logical flag to reindex the joined resource (TRUE) upon joining. The default is **no** reindexing.

Parameters

in	*label*	label Label for the joined food resource, if absent set to 'tmp_object'.

Returns

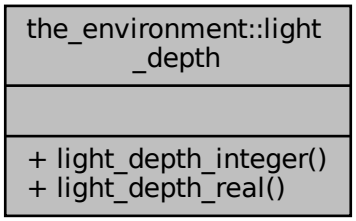
A collapsed food resource joining the input array.

Definition at line 7914 of file m_env.f90.
The documentation for this interface was generated from the following file:

- m_env.f90

9.41 the_environment::light_depth Interface Reference

Calculate *underwater background irradiance* at specific depth.
Collaboration diagram for the_environment::light_depth:



Public Member Functions

- real(srp) function light_depth_integer (depth, surface_light, is_stochastic)

  Calculate underwater light at specific depth given specific surface light.
- real(srp) function light_depth_real (depth, surface_light, is_stochastic)

  Calculate underwater light at specific depth given specific surface light.

9.41.1 Detailed Description

Calculate *underwater background irradiance* at specific depth.
Underwater light is attenuated following Beer's law,

$$E_b(z,t) = L_t e^{-Kz},$$

where
Definition at line 715 of file m_env.f90.

9.41.2 Member Function/Subroutine Documentation

9.41.2.1 light_depth_integer()

```
real(srp) function the_environment::light_depth::light_depth_integer (
            integer, intent(in) depth,
            real(srp), intent(in), optional surface_light,
            logical, intent(in), optional is_stochastic )
```

Calculate underwater light at specific depth given specific surface light.
Underwater light is attenuated following Beer's law,

$$E_b(z,t) = L_t e^{-Kz},$$

where $E_b(z,t)$ is background irradiance at depth z at time t and K is the attenuation coefficient for downwelling irradiance. The value of K in the old code was set very high to allow the vertical dynamics to take place within 30 depth cells.

Returns

Eb background irradiance at specific depth.

Parameters

in	*depth*	The integer depth horizon where we get background
in	*surface_light*	Irradiance at the surface, normally calculated at specific time point of the model with the the_environment::light_surface() function. If this parameter is absent, surface light at the current time step is obtained. The time step in such case is obtained from commondata::global_time_step_model_current.
in	*is_stochastic*	stochastic indicator for the surface light in the_environment::light_surface() function. If this parameter is absent, the default commondata::daylight_stochastic parameter value is used.

Note

Note that this function accepts **integer** depth, a separate function should be used for physical real type depth.

Note that it is an elemental function that accepts both scalar and array parameters.

Definition at line 5507 of file m_env.f90.

9.41.2.2 light_depth_real()

```
real(srp) function the_environment::light_depth::light_depth_real (
            real(srp), intent(in) depth,
            real(srp), intent(in), optional surface_light,
            logical, intent(in), optional is_stochastic )
```

Calculate underwater light at specific depth given specific surface light.
Underwater light is attenuated following Beer's law,

$$E_b(z,t) = L_t e^{-Kz},$$

where $E_b(z,t)$ is background irradiance at depth z at time t and K is the attenuation coefficient for downwelling irradiance.

Returns

Eb background irradiance at specific depth.

Parameters

in	*depth*	The integer depth horizon where we get background.

Parameters

in	*surface_light*	Irradiance at the surface, normally calculated at specific time point of the model with the the_environment::light_surface() function. If this parameter is absent, surface light at the current time step is obtained. The time step in such case is obtained from commondata::global_time_step_model_current.
in	*is_stochastic*	stochastic indicator for the surface light in the_environment::light_surface() function. If this parameter is absent, the default commondata::daylight_stochastic parameter value is used.

Note

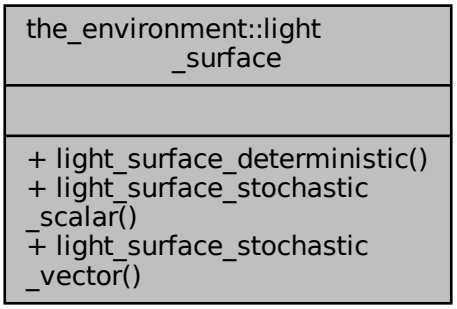
Note that this function accepts **real** depth.

Definition at line 5582 of file m_env.f90.
The documentation for this interface was generated from the following file:

- m_env.f90

9.42 the_environment::light_surface Interface Reference

Calculate *surface light* intensity (that is subject to diel variation) for specific time step of the model. Irradiance can be *stochastic* if an optionallogical `stochastic` flag is set to `TRUE`.
Collaboration diagram for the_environment::light_surface:



Public Member Functions

- elemental real(srp) function light_surface_deterministic (tstep)

  Calculate deterministic surface light at specific time step of the model. Light (`surlig`) is calculated from a sine function. Light intensity just beneath the surface is modelled by assuming a 50 % loss by scattering at the surface:
- real(srp) function light_surface_stochastic_scalar (tstep, is_stochastic)

  Calculate stochastic surface light at specific time step of the model. Light (`surlig`) is calculated from a sine function. Light intensity just beneath the surface is modelled by assuming a 50 % loss by scattering at the surface:
- real(srp) function, dimension(size(tstep)) light_surface_stochastic_vector (tstep, is_stochastic)

  Calculate stochastic surface light at specific time step of the model.

9.42.1 Detailed Description

Calculate *surface light* intensity (that is subject to diel variation) for specific time step of the model. Irradiance can be *stochastic* if an optionallogical `stochastic` flag is set to `TRUE`.
Light (`surlig`) is calculated from a sine function. Light intensity just beneath the surface is modelled by assuming a 50 % loss by scattering at the surface:

$$L_t = L_{max} 0.5 sin(\pi dt/\Omega).$$

Usage:

- deterministic:
  ```
  surface_light(1)
  ```
- stochastic:
  ```
  surface_light(1,yes)
  ```

Note

Note that it is impossible to do a simple single whole-elemental implementation for this function as `random_number` is *never pure* but elemental can only work with all pure functions.

Definition at line 702 of file m_env.f90.

9.42.2 Member Function/Subroutine Documentation

9.42.2.1 light_surface_deterministic()

```
elemental real(srp) function the_environment::light_surface::light_surface_deterministic (
            integer, intent(in), optional tstep )
```

Calculate deterministic surface light at specific time step of the model. Light (`surlig`) is calculated from a sine function. Light intensity just beneath the surface is modelled by assuming a 50 % loss by scattering at the surface:

$$L_t = L_{max} 0.5 sin(\pi dt/\Omega)$$

.

Returns

surface light intensity.

Parameters

tstep	time step of the model, limited by maximum commondata::lifespan.

Note

This is a deterministic version. Code for wxMaxima for quickcalc:
```
surlig(a, span) := 500*0.5*(1.01+sin(3.14*2.*50*a/(1.*span)));
surlig(a, span) := 500*0.5*(1.01+sin(3.14*2.*50*a/span));
wxplot2d(surlig(a, 14000), [a,0, 1400]);
```

Note that this is an elemental function that accepts both scalar and array parameter.

Definition at line 5359 of file m_env.f90.

9.42.2.2 light_surface_stochastic_scalar()

```
real(srp) function the_environment::light_surface::light_surface_stochastic_scalar (
            integer, intent(in), optional tstep,
            logical, intent(in) is_stochastic )
```

Calculate stochastic surface light at specific time step of the model. Light (`surlig`) is calculated from a sine function. Light intensity just beneath the surface is modelled by assuming a 50 % loss by scattering at the surface:

$$L_t = L_{max} 0.5 sin(\pi dt/\Omega)$$

. This deterministic value sets the *mean* for the stochastic final value, which is Gaussian with CV equal to `DAYLIGHT_CV`.

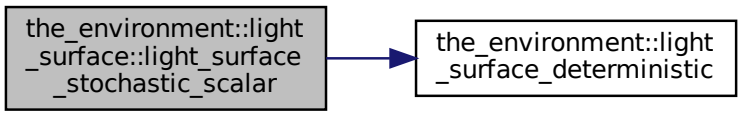
Returns

surface light intensity

Parameters

tstep	time step of the model, limited by maximum commondata::lifespan.
is_stochastic	logical indicator for stochastic light intensity if TRUE, then Gaussian stochastic version is used, if FALSE, deterministic is used. Code for wxMaxima for quickcalc: `surlig(a, span) := 500*0.5*(1.01+sin(3.14*2.*50*a/(1.*span))); wxplot2d(surlig(a, 14000), [a,0, 1400]);`

Definition at line 5399 of file m_env.f90.
Here is the call graph for this function:



9.42.2.3 light_surface_stochastic_vector()

```
real(srp) function, dimension(size(tstep)) the_environment::light_surface::light_surface_↩
stochastic_vector (
            integer, dimension(:), intent(in) tstep,
            logical, intent(in) is_stochastic )
```

Calculate stochastic surface light at specific time step of the model.

Parameters

tstep	time step of the model, limited by maximum commondata::lifespan.

Returns

surface light intensity.

Parameters

is_stochastic	logical indicator for stochastic light intensity if TRUE, then Gaussian stochastic version is used, if FALSE, deterministic is used.

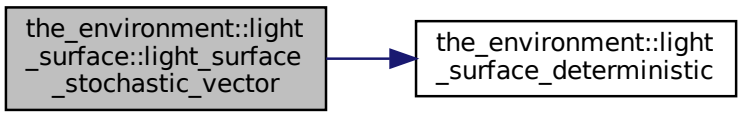
Note

This function accepts vector arguments.

Warning

Note that the `tstep` array parameter is *mandatory* here (otherwise the generic interface is ambiguous).

Definition at line 5447 of file m_env.f90.
Here is the call graph for this function:

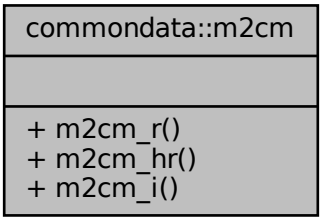


The documentation for this interface was generated from the following file:

- m_env.f90

9.43 commondata::m2cm Interface Reference

Convert m to cm.
Collaboration diagram for commondata::m2cm:



Public Member Functions

- elemental real(srp) function m2cm_r (value_m)

  Convert m to cm.
- elemental real(hrp) function m2cm_hr (value_m)

  Convert m to cm.
- elemental real(srp) function m2cm_i (value_m)

  Convert m to cm.

9.43.1 Detailed Description

Convert m to cm.
This is needed because some of the the sizes are expressed in cm but certain functions (e.g. visual range estimator SRGETR) require parameters in m. Therefore, conversion back from visual range function should use these functions.
Definition at line 5319 of file m_common.f90.

9.43.2 Member Function/Subroutine Documentation

9.43.2.1 m2cm_r()

```
elemental real(srp) function commondata::m2cm::m2cm_r (
            real(srp), intent(in) value_m )
```

Convert m to cm.

Parameters

value_cm	value in cm

Returns

value in m

Note

This version gets real type argument.

Definition at line 5592 of file m_common.f90.

9.43.2.2 m2cm_hr()

```
elemental real(hrp) function commondata::m2cm::m2cm_hr (
            real(hrp), intent(in) value_m )
```

Convert m to cm.

Returns

value in m

Parameters

value_cm	value in cm

Note

This version uses the **high** HRP numerical precision model.

Definition at line 5606 of file m_common.f90.

9.43.2.3 m2cm_i()

```
elemental real(srp) function commondata::m2cm::m2cm_i (
            integer, intent(in) value_m )
```

Convert m to cm.

Parameters

value_cm	value in cm

Returns

value in m

Note

This version gets integer argument (albeit returns real).

Definition at line 5620 of file m_common.f90.
The documentation for this interface was generated from the following file:

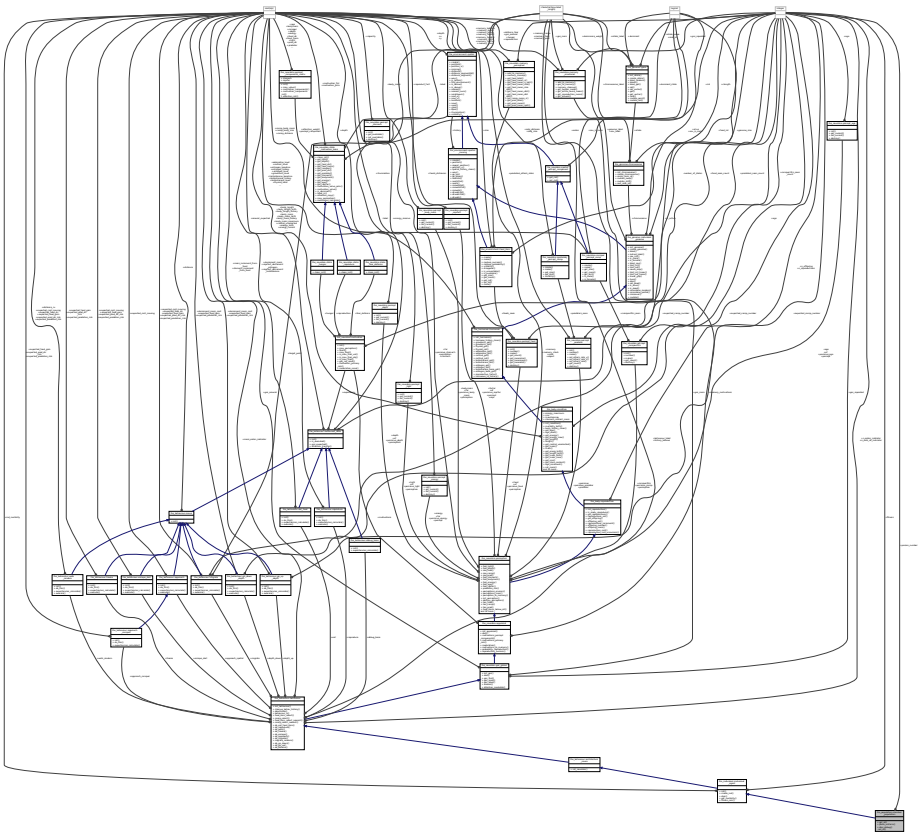
- m_common.f90

9.44 the_population::member_population Type Reference

Definition of individual member of a population.

Inheritance diagram for the_population::member_population:

Collaboration diagram for the_population::member_population:



Public Member Functions

- procedure, public get_id => get_individual_id

  Get integer ID number to individual member of the population object. See `the_population::get_individual_id()`.
- procedure, public place_uniform => individ_posit_in_environ_uniform

  Places the individual agent, a member of the population, within a specific environment at random with a ***uniform*** *distribution. See* `the_population::individ_posit_in_environ_uniform()`.
- procedure, public dies_debug => genome_individual_set_dead_non_pure

  Set the individual to be ***dead****. See* `the_population::genome_individual_set_dead_non_pure()`.

Public Attributes

- integer person_number

Private Member Functions

- procedure, private set_id => set_individual_id

  Set integer ID number to individual member of the population object. See `the_population::set_individual_id()`.

9.44.1 Detailed Description

Definition of individual member of a population.
Add an additional object, a member of the population. It is the`INDIVIDUAL_AGENT` adding an (unique) integer ID: `person_number`. Here we also have two functions for setting and retrieving the IDs. We do not init members of the population using a single init function, Instead, init normally the INDIVIDUAL_AGENT object and for the MEMBER_POPULATION just set_id. This is because we cannot set id for isolated subject, only as a member of population.
Definition at line 36 of file m_popul.f90.

9.44.2 Member Function/Subroutine Documentation

9.44.2.1 set_id()

`procedure, private the_population::member_population::set_id [private]`
Set integer ID number to individual member of the population object. See `the_population::set_individual_id()`.
Definition at line 42 of file m_popul.f90.

9.44.2.2 get_id()

`procedure, public the_population::member_population::get_id`
Get integer ID number to individual member of the population object. See `the_population::get_individual_id()`.
Definition at line 45 of file m_popul.f90.

9.44.2.3 place_uniform()

`procedure, public the_population::member_population::place_uniform`
Places the individual agent, a member of the population, within a specific environment at random with a **uniform** distribution. See `the_population::individ_posit_in_environ_uniform()`.
Definition at line 49 of file m_popul.f90.

9.44.2.4 dies_debug()

`procedure, public the_population::member_population::dies_debug`
Set the individual to be **dead**. See `the_population::genome_individual_set_dead_non_pure()`.
Definition at line 52 of file m_popul.f90.

9.44.3 Member Data Documentation

9.44.3.1 person_number

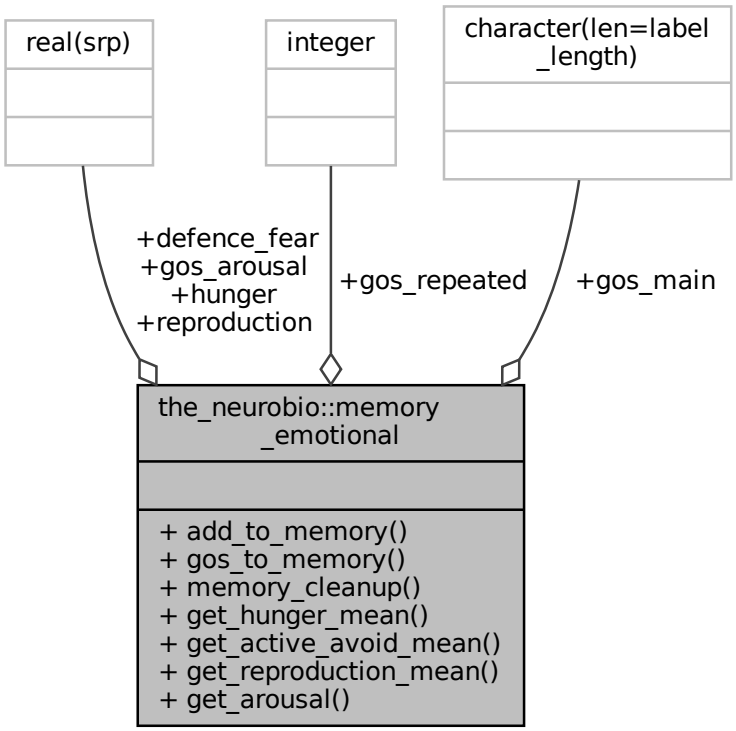
`integer the_population::member_population::person_number`
Definition at line 37 of file m_popul.f90.
The documentation for this type was generated from the following file:

- m_popul.f90

9.45 the_neurobio::memory_emotional Type Reference

Individual motivation/emotion memory stack, a memory component that saves the values of the **final motivations** at previous time steps of the model. Not whole state (`STATE_`) objects are saved for simplicity. `add_to_history` is used in unmodified form. Decision making can make use of this emotional memory stack.

Collaboration diagram for the_neurobio::memory_emotional:



Public Member Functions

- procedure, public add_to_memory => emotional_memory_add_to_stack

  Add emotional components into the memory stack. See the_neurobio::emotional_memory_add_to_stack().
- procedure, public gos_to_memory => emotional_memory_add_gos_to_stack

  Add the current GOS label or/and arousal value and/or arousal repeat count into the emotional memory stack. See the_neurobio::emotional_memory_add_gos_to_stack().
- procedure, public memory_cleanup => emotional_memory_cleanup_stack

  Cleanup and destroy the emotional memory stack. See the_neurobio::emotional_memory_cleanup_stack().
- procedure, public get_hunger_mean => emotional_memory_hunger_get_mean

  Get the average value of the hunger motivation state within the whole emotional memory stack. See the_neurobio::emotional_memory_hunger_get_mean().
- procedure, public get_active_avoid_mean => emotional_memory_actve_avoid_get_mean

  Get the average value of the fear state motivation state within the whole emotional memory stack. See the_neurobio::emotional_memory_actve_avoid_get_mean().
- procedure, public get_reproduction_mean => emotional_memory_reproduct_get_mean

  Get the average value of the reproductive motivation state within the whole emotional memory stack. See the_neurobio::emotional_memory_reproduct_get_mean().
- procedure, public get_arousal => emotional_memory_arousal_mean

  Get the average value of the GOS arousal within the whole emotional memory stack. See the_neurobio::emotional_memory_a

Public Attributes

- real(srp), dimension(history_size_motivation) hunger
- real(srp), dimension(history_size_motivation) defence_fear
- real(srp), dimension(history_size_motivation) reproduction
- character(len=label_length), dimension(history_size_motivation) gos_main

  Memory also includes a component for the global organismic state (GOS).
- real(srp), dimension(history_size_motivation) gos_arousal

  Memory also includes the motivation level that has resulted in the current GOS. This is a memory for the arousal. Although doubles one of the basic motivations (hunger etc.), but here for convenience.
- integer, dimension(history_size_motivation) gos_repeated

  Memory also includes the GOS repeat counter, this is the number of times that the same GOS state is repeated. See `the_neurobio::gos_global` for implementation details.

9.45.1 Detailed Description

Individual motivation/emotion memory stack, a memory component that saves the values of the **final motivations** at previous time steps of the model. Not whole state (`STATE_`) objects are saved for simplicity. `add_to_history` is used in unmodified form. Decision making can make use of this emotional memory stack.
Definition at line 1166 of file m_neuro.f90.

9.45.2 Member Function/Subroutine Documentation

9.45.2.1 add_to_memory()

`procedure, public the_neurobio::memory_emotional::add_to_memory`
Add emotional components into the memory stack. See `the_neurobio::emotional_memory_add_to_stack()`.
Definition at line 1185 of file m_neuro.f90.

9.45.2.2 gos_to_memory()

`procedure, public the_neurobio::memory_emotional::gos_to_memory`
Add the current GOS label or/and arousal value and/or arousal repeat count into the emotional memory stack. See `the_neurobio::emotional_memory_add_gos_to_stack()`.
Definition at line 1189 of file m_neuro.f90.

9.45.2.3 memory_cleanup()

`procedure, public the_neurobio::memory_emotional::memory_cleanup`
Cleanup and destroy the emotional memory stack. See `the_neurobio::emotional_memory_cleanup_stack()`.
Definition at line 1192 of file m_neuro.f90.

9.45.2.4 get_hunger_mean()

`procedure, public the_neurobio::memory_emotional::get_hunger_mean`
Get the average value of the hunger motivation state within the whole emotional memory stack. See `the_neurobio::emotional_memory_hunger_get_mean()`.
Definition at line 1196 of file m_neuro.f90.

9.45.2.5 get_active_avoid_mean()

procedure, public the_neurobio::memory_emotional::get_active_avoid_mean
Get the average value of the fear state motivation state within the whole emotional memory stack. See the_neurobio::emotional_memory_actve_avoid_get_mean().
Definition at line 1201 of file m_neuro.f90.

9.45.2.6 get_reproduction_mean()

procedure, public the_neurobio::memory_emotional::get_reproduction_mean
Get the average value of the reproductive motivation state within the whole emotional memory stack. See the_neurobio::emotional_memory_reproduct_get_mean().
Definition at line 1206 of file m_neuro.f90.

9.45.2.7 get_arousal()

procedure, public the_neurobio::memory_emotional::get_arousal
Get the average value of the GOS arousal within the whole emotional memory stack. See the_neurobio::emotional_memory
Definition at line 1211 of file m_neuro.f90.

9.45.3 Member Data Documentation

9.45.3.1 hunger

real(srp), dimension(history_size_motivation) the_neurobio::memory_emotional::hunger
Definition at line 1167 of file m_neuro.f90.

9.45.3.2 defence_fear

real(srp), dimension(history_size_motivation) the_neurobio::memory_emotional::defence_fear
Definition at line 1168 of file m_neuro.f90.

9.45.3.3 reproduction

real(srp), dimension(history_size_motivation) the_neurobio::memory_emotional::reproduction
Definition at line 1169 of file m_neuro.f90.

9.45.3.4 gos_main

character(len=label_length), dimension(history_size_motivation) the_neurobio::memory_emotional↩
::gos_main
Memory also includes a component for the global organismic state (GOS).

Note

Note that GOS cannot be determined at the APPRAISAL level, is updated later, so we may need a separate add-to-memory function.

Definition at line 1173 of file m_neuro.f90.

9.45.3.5 gos_arousal

`real(srp), dimension(history_size_motivation) the_neurobio::memory_emotional::gos_arousal`
Memory also includes the motivation level that has resulted in the current GOS. This is a memory for the arousal. Although doubles one of the basic motivations (hunger etc.), but here for convenience.
Definition at line 1177 of file m_neuro.f90.

9.45.3.6 gos_repeated

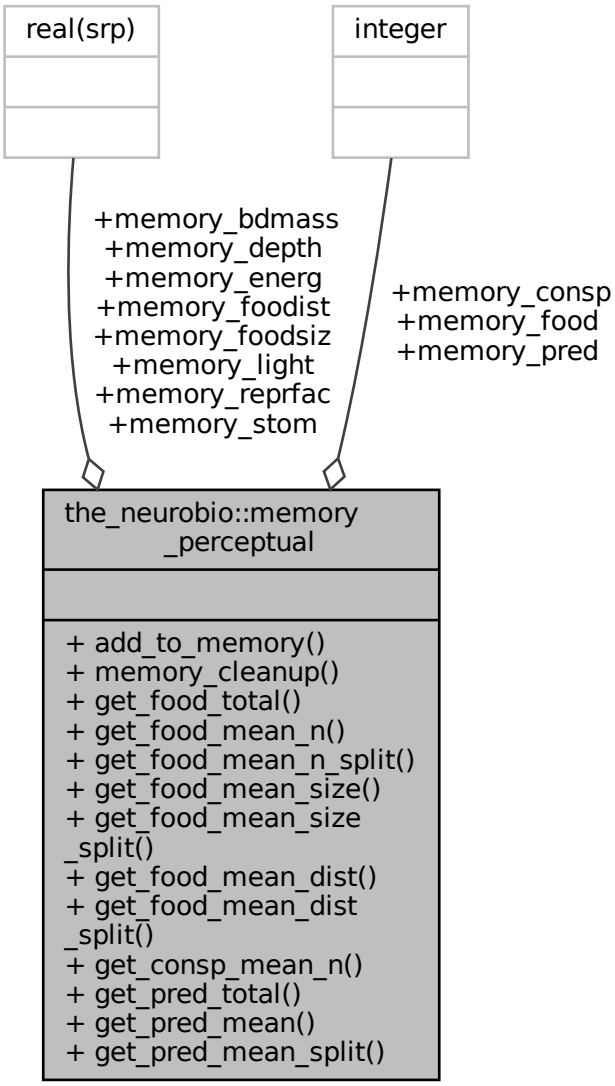
`integer, dimension(history_size_motivation) the_neurobio::memory_emotional::gos_repeated`
Memory also includes the GOS repeat counter, this is the number of times that the same GOS state is repeated. See `the_neurobio::gos_global` for implementation details.
Definition at line 1181 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

9.46 the_neurobio::memory_perceptual Type Reference

Individual perception memory(history) stack, a memory component that saves perception values at previous time steps of the model. Not whole perception objects are saved for simplicity, only the most important parameters, integer and real types so commondata::add_to_history() can be used in unmodified form. Decision making can make use of this memory stack.

Collaboration diagram for the_neurobio::memory_perceptual:



Public Member Functions

- procedure, public add_to_memory => percept_memory_add_to_stack

  Add perception components into the memory stack. See `the_neurobio::percept_memory_add_to_stack()`.
- procedure, public memory_cleanup => percept_memory_cleanup_stack

  Cleanup and destroy the perceptual memory stack. See `the_neurobio::percept_memory_cleanup_stack()`.
- procedure, public get_food_total => percept_memory_food_get_total

  Get the total number of food items within the whole perceptual memory stack. See `the_neurobio::percept_memory_food_get`
- procedure, public get_food_mean_n => percept_memory_food_get_mean_n

  Get the average number of food items per single time step within the whole perceptual memory stack. See `the_neurobio::percept_memory_food_get_mean_n()`.
- procedure, public get_food_mean_n_split => percept_memory_food_mean_n_split

*Get the **average number** of food items per single time step within the perceptual memory stack, split to the first (older) and second (newer) parts. The whole memory stack ('sample') is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it. See `the_neurobio::percept_memory_food_mean_n_split()`.*

- procedure, public get_food_mean_size =>percept_memory_food_get_mean_size

  Get the average size of food item per single time step within the whole perceptual memory stack. See `the_neurobio::percept_memory_food_get_mean_size()`.

- procedure, public get_food_mean_size_split => percept_memory_food_mean_size_split

  *Get the **average size** of food items per single time step within the perceptual memory stack, split to the first (older) and second(newer) parts. The whole memory stack 'sample' is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it. See `the_neurobio::percept_memory_food_mean_size_split()`.*

- procedure, public get_food_mean_dist => percept_memory_food_get_mean_dist

  *Get the **average distance** to food item per single time step within the whole perceptual memory stack. See `the_neurobio::percept_memory_food_get_mean_dist()`.*

- procedure, public get_food_mean_dist_split => percept_memory_food_mean_dist_split

  *Get the **average distance** to food items per single time step within the perceptual memory stack, split to the first (older) and second(newer) parts. The whole memory stack 'sample' is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it. See `the_neurobio::percept_memory_food_mean_dist_spli`*

- procedure, public get_consp_mean_n => percept_memory_consp_get_mean_n

  *Get the **average number** of conspecifics per single time step within the whole perceptual memory stack. See `the_neurobio::percept_memory_consp_get_mean_n()`.*

- procedure, public get_pred_total => percept_memory_predators_get_total

  Get the total number of predators within the whole perceptual memory stack. See `the_neurobio::percept_memory_predator`

- procedure, public get_pred_mean => percept_memory_predators_get_mean

  Get the average number of predators per single time step within the whole perceptual memory stack. See `the_neurobio::percept_memory_predators_get_mean()`.

- procedure, public get_pred_mean_split => percept_memory_predators_mean_split

  *Get the **average number** of predators per single time step within the perceptual memory stack, split to the first (older) and second(newer) parts. The whole memory stack ('sample') is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it. See `the_neurobio::percept_memory_predators_mean_split()`.*

Public Attributes

- real(srp), dimension(history_size_perception) memory_light

  *Memory for **light**.*

- real(srp), dimension(history_size_perception) memory_depth

  *Memory for **depth**.*

- integer, dimension(history_size_perception) memory_food

  *Memory for **number of food items** seen (in perception).*

- real(srp), dimension(history_size_perception) memory_foodsiz

  *Memory for **mean size of food items** seen (in perception).*

- real(srp), dimension(history_size_perception) memory_foodist

  *Memory for **mean distance to the food items** seen (in perception).*

- integer, dimension(history_size_perception) memory_consp

  *Memory for **number of conspecifics** seen.*

- integer, dimension(history_size_perception) memory_pred

  *Memory for **number of predators**.*

- real(srp), dimension(history_size_perception) memory_stom

  *Memory for **stomach contents**.*

- real(srp), dimension(history_size_perception) memory_bdmass

  *Memory for **body mass**.*

- real(srp), dimension(history_size_perception) memory_energ

  *Memory for **energy reserves**.*

- real(srp), dimension(history_size_perception) memory_reprfac

  *Memory for **reproductive factor** values.*

9.46.1 Detailed Description

Individual perception memory(history) stack, a memory component that saves perception values at previous time steps of the model. Not whole perception objects are saved for simplicity, only the most important parameters, integer and real types so commondata::add_to_history() can be used in unmodified form. Decision making can make use of this memory stack.

Note

Note that age perception `the_neurobio::percept_age` is **not saved** in memory stack as it is trivial to get/predict.

Definition at line 447 of file m_neuro.f90.

9.46.2 Member Function/Subroutine Documentation

9.46.2.1 add_to_memory()

`procedure, public the_neurobio::memory_perceptual::add_to_memory`
Add perception components into the memory stack. See `the_neurobio::percept_memory_add_to_stack()`.
Definition at line 473 of file m_neuro.f90.

9.46.2.2 memory_cleanup()

`procedure, public the_neurobio::memory_perceptual::memory_cleanup`
Cleanup and destroy the perceptual memory stack. See `the_neurobio::percept_memory_cleanup_stack()`.
Definition at line 476 of file m_neuro.f90.

9.46.2.3 get_food_total()

`procedure, public the_neurobio::memory_perceptual::get_food_total`
Get the total number of food items within the whole perceptual memory stack. See `the_neurobio::percept_memory_food_`
Definition at line 479 of file m_neuro.f90.

9.46.2.4 get_food_mean_n()

`procedure, public the_neurobio::memory_perceptual::get_food_mean_n`
Get the average number of food items per single time step within the whole perceptual memory stack. See `the_neurobio::percept_memory_food_get_mean_n()`.
Definition at line 483 of file m_neuro.f90.

9.46.2.5 get_food_mean_n_split()

`procedure, public the_neurobio::memory_perceptual::get_food_mean_n_split`
Get the **average number** of food items per single time step within the perceptual memory stack, split to the first (older) and second (newer) parts. The whole memory stack ('sample') is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it. See `the_neurobio::percept_memory_food_mean_n_split()`.
Definition at line 490 of file m_neuro.f90.

9.46.2.6 get_food_mean_size()

procedure, public the_neurobio::memory_perceptual::get_food_mean_size
Get the average size of food item per single time step within the whole perceptual memory stack. See the_neurobio::percept_memory_food_get_mean_size().
Definition at line 495 of file m_neuro.f90.

9.46.2.7 get_food_mean_size_split()

procedure, public the_neurobio::memory_perceptual::get_food_mean_size_split
Get the **average size** of food items per single time step within the perceptual memory stack, split to the first (older) and second(newer) parts. The whole memory stack 'sample' is split by the split_val parameter and two means are calculated: before the split_val and after it. See the_neurobio::percept_memory_food_mean_size_split()
Definition at line 501 of file m_neuro.f90.

9.46.2.8 get_food_mean_dist()

procedure, public the_neurobio::memory_perceptual::get_food_mean_dist
Get the **average distance** to food item per single time step within the whole perceptual memory stack. See the_neurobio::percept_memory_food_get_mean_dist().
Definition at line 506 of file m_neuro.f90.

9.46.2.9 get_food_mean_dist_split()

procedure, public the_neurobio::memory_perceptual::get_food_mean_dist_split
Get the **average distance** to food items per single time step within the perceptual memory stack, split to the first (older) and second(newer) parts. The whole memory stack 'sample' is split by the split_val parameter and two means are calculated: before the split_val and after it. See the_neurobio::percept_memory_food_mean_dist_split().
Definition at line 513 of file m_neuro.f90.

9.46.2.10 get_consp_mean_n()

procedure, public the_neurobio::memory_perceptual::get_consp_mean_n
Get the **average number** of conspecifics per single time step within the whole perceptual memory stack. See the_neurobio::percept_memory_consp_get_mean_n().
Definition at line 518 of file m_neuro.f90.

9.46.2.11 get_pred_total()

procedure, public the_neurobio::memory_perceptual::get_pred_total
Get the total number of predators within the whole perceptual memory stack. See the_neurobio::percept_memory_predat
Definition at line 521 of file m_neuro.f90.

9.46.2.12 get_pred_mean()

procedure, public the_neurobio::memory_perceptual::get_pred_mean
Get the average number of predators per single time step within the whole perceptual memory stack. See the_neurobio::percept_memory_predators_get_mean().
Definition at line 525 of file m_neuro.f90.

9.46.2.13 get_pred_mean_split()

`procedure, public the_neurobio::memory_perceptual::get_pred_mean_split`
Get the **average number** of predators per single time step within the perceptual memory stack, split to the first (older) and second(newer) parts. The whole memory stack ('sample') is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it. See the_neurobio::percept_memory_predators_mean_sp
Definition at line 531 of file m_neuro.f90.

9.46.3 Member Data Documentation

9.46.3.1 memory_light

`real(srp), dimension(history_size_perception) the_neurobio::memory_perceptual::memory_light`
Memory for **light**.
Definition at line 449 of file m_neuro.f90.

9.46.3.2 memory_depth

`real(srp), dimension(history_size_perception) the_neurobio::memory_perceptual::memory_depth`
Memory for **depth**.
Definition at line 451 of file m_neuro.f90.

9.46.3.3 memory_food

`integer, dimension(history_size_perception) the_neurobio::memory_perceptual::memory_food`
Memory for **number of food items** seen (in perception).
Definition at line 453 of file m_neuro.f90.

9.46.3.4 memory_foodsiz

`real(srp), dimension(history_size_perception) the_neurobio::memory_perceptual::memory_foodsiz`
Memory for **mean size of food items** seen (in perception).
Definition at line 455 of file m_neuro.f90.

9.46.3.5 memory_foodist

`real(srp), dimension(history_size_perception) the_neurobio::memory_perceptual::memory_foodist`
Memory for **mean distance to the food items** seen (in perception).
Definition at line 457 of file m_neuro.f90.

9.46.3.6 memory_consp

`integer, dimension(history_size_perception) the_neurobio::memory_perceptual::memory_consp`
Memory for **number of conspecifics** seen.
Definition at line 459 of file m_neuro.f90.

9.46.3.7 memory_pred

`integer, dimension(history_size_perception) the_neurobio::memory_perceptual::memory_pred`
Memory for **number of predators**.
Definition at line 461 of file m_neuro.f90.

9.46.3.8 memory_stom

`real(srp), dimension(history_size_perception) the_neurobio::memory_perceptual::memory_stom`
Memory for **stomach contents**.
Definition at line 463 of file m_neuro.f90.

9.46.3.9 memory_bdmass

`real(srp), dimension(history_size_perception) the_neurobio::memory_perceptual::memory_bdmass`
Memory for **body mass**.
Definition at line 465 of file m_neuro.f90.

9.46.3.10 memory_energ

`real(srp), dimension(history_size_perception) the_neurobio::memory_perceptual::memory_energ`
Memory for **energy reserves**.
Definition at line 467 of file m_neuro.f90.

9.46.3.11 memory_reprfac

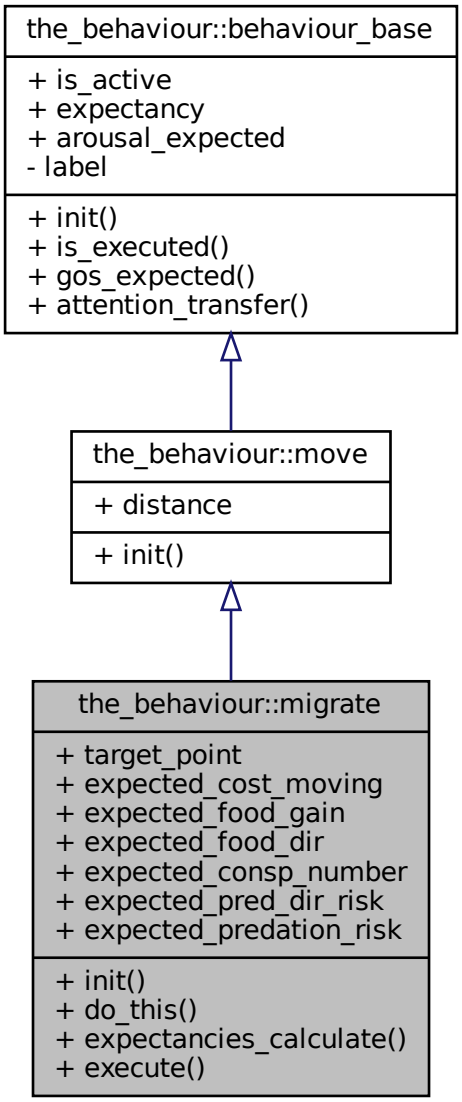
`real(srp), dimension(history_size_perception) the_neurobio::memory_perceptual::memory_reprfac`
Memory for **reproductive factor** values.
Definition at line 469 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

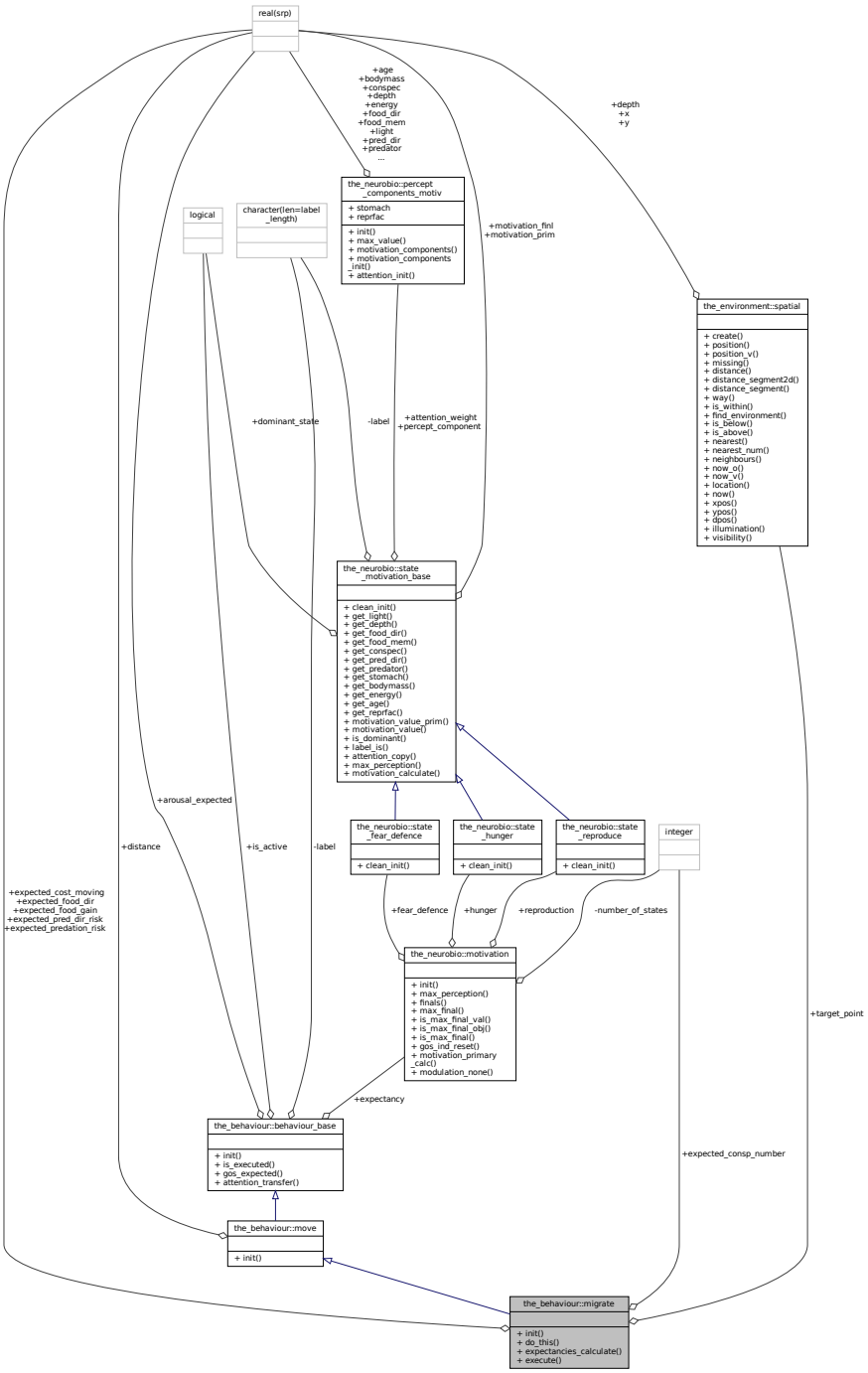
9.47 the_behaviour::migrate Type Reference

Migrate is move quickly directing to the other habitat

Inheritance diagram for the_behaviour::migrate:



Collaboration diagram for the_behaviour::migrate:



Public Member Functions

- procedure, public init => migrate_init_zero

  *Initialise the **migrate** behaviour component to a zero state. See* `the_behaviour::migrate_init_zero()`.
- procedure, public do_this => migrate_do_this

  The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here `MIGRATE`*). See* `the_behaviour::migrate_do_this()`.

- procedure, public expectancies_calculate => migrate_motivations_expect

  the_behaviour::migrate::expectancies_calculate() (re)calculates motivations from fake expected perceptions following from the procedure migrate::do_this() => the_behaviour::migrate_do_this(). See the_behaviour::migrate_motivations_expect().
- procedure, public execute => migrate_do_execute

  Execute this behaviour component "migrate" by this_agent agent. See the_behaviour::migrate_do_execute().

Public Attributes

- type(spatial) target_point

  Target point (with offset) for migration into the target environment.
- real(srp) expected_cost_moving

  The body mass cost of movement; depends on the distance.
- real(srp) expected_food_gain

  The expected food gain (for body mass increment).
- real(srp) expected_food_dir

  The expected direct food perception in the novel target habitat.
- integer expected_consp_number

  The expected number of conspecifics at the layer below. This value is based on the number of conspecifics below the agent's current horizon.
- real(srp) expected_pred_dir_risk

  The expected direct predation risk is zero for random walk.
- real(srp) expected_predation_risk

  The expected general predation risk, i.e. the risk depending on the current number of predators in both the perception and memory stack.

9.47.1 Detailed Description

Migrate is move quickly directing to the other habitat
Definition at line 355 of file m_behav.f90.

9.47.2 Member Function/Subroutine Documentation

9.47.2.1 init()

`procedure, public the_behaviour::migrate::init`
Initialise the **migrate** behaviour component to a zero state. See the_behaviour::migrate_init_zero().
Definition at line 375 of file m_behav.f90.

9.47.2.2 do_this()

`procedure, public the_behaviour::migrate::do_this`
The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here MIGRATE). See the_behaviour::migrate_do_this().
Definition at line 382 of file m_behav.f90.

9.47.2.3 expectancies_calculate()

`procedure, public the_behaviour::migrate::expectancies_calculate`
`the_behaviour::migrate::expectancies_calculate()` (re)calculates motivations from fake expected perceptions following from the procedure `migrate::do_this()` => `the_behaviour::migrate_do_this()`.
See `the_behaviour::migrate_motivations_expect()`.
Definition at line 387 of file m_behav.f90.

9.47.2.4 execute()

`procedure, public the_behaviour::migrate::execute`
Execute this behaviour component "migrate" by `this_agent` agent. See `the_behaviour::migrate_do_execute()`.
Definition at line 390 of file m_behav.f90.

9.47.3 Member Data Documentation

9.47.3.1 target_point

`type(spatial) the_behaviour::migrate::target_point`
Target point (with offset) for migration into the target environment.
Definition at line 357 of file m_behav.f90.

9.47.3.2 expected_cost_moving

`real(srp) the_behaviour::migrate::expected_cost_moving`
The body mass cost of movement; depends on the distance.
Definition at line 359 of file m_behav.f90.

9.47.3.3 expected_food_gain

`real(srp) the_behaviour::migrate::expected_food_gain`
The expected food gain (for body mass increment).
Definition at line 361 of file m_behav.f90.

9.47.3.4 expected_food_dir

`real(srp) the_behaviour::migrate::expected_food_dir`
The expected direct food perception in the novel target habitat.
Definition at line 363 of file m_behav.f90.

9.47.3.5 expected_consp_number

`integer the_behaviour::migrate::expected_consp_number`
The expected number of conspecifics at the layer below. This value is based on the number of conspecifics below the agent's current horizon.
Definition at line 366 of file m_behav.f90.

9.47.3.6 expected_pred_dir_risk

`real(srp) the_behaviour::migrate::expected_pred_dir_risk`
The expected direct predation risk is zero for random walk.
Definition at line 368 of file m_behav.f90.

9.47.3.7 expected_predation_risk

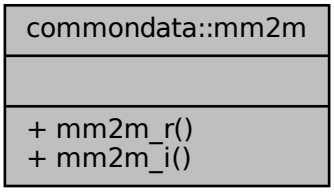
```
real(srp) the_behaviour::migrate::expected_predation_risk
```

The expected general predation risk, i.e. the risk depending on the current number of predators in both the perception and memory stack.
Definition at line 371 of file m_behav.f90.
The documentation for this type was generated from the following file:

- m_behav.f90

9.48 commondata::mm2m Interface Reference

Convert mm to m.
Collaboration diagram for commondata::mm2m:



Public Member Functions

- elemental real(srp) function mm2m_r (value_mm)

  Convert mm to m.
- elemental real(srp) function mm2m_i (value_mm)

  Convert mm to m.

9.48.1 Detailed Description

Convert mm to m.

Note

This is a similar functions(s) as above sm2m, but converting mm.

Definition at line 5329 of file m_common.f90.

9.48.2 Member Function/Subroutine Documentation

9.48.2.1 mm2m_r()

```
elemental real(srp) function commondata::mm2m::mm2m_r (
            real(srp), intent(in) value_mm )
```

Convert mm to m.

Parameters

value_cm	value in cm

Returns

value in m

Note

This version gets real type argument.

Definition at line 5634 of file m_common.f90.

9.48.2.2 mm2m_i()

```
elemental real(srp) function commondata::mm2m::mm2m_i (
             integer, intent(in) value_mm )
```

Convert mm to m.

Parameters

value_cm	value in cm

Returns

value in m

Note

This version gets integer argument (albeit returns real).

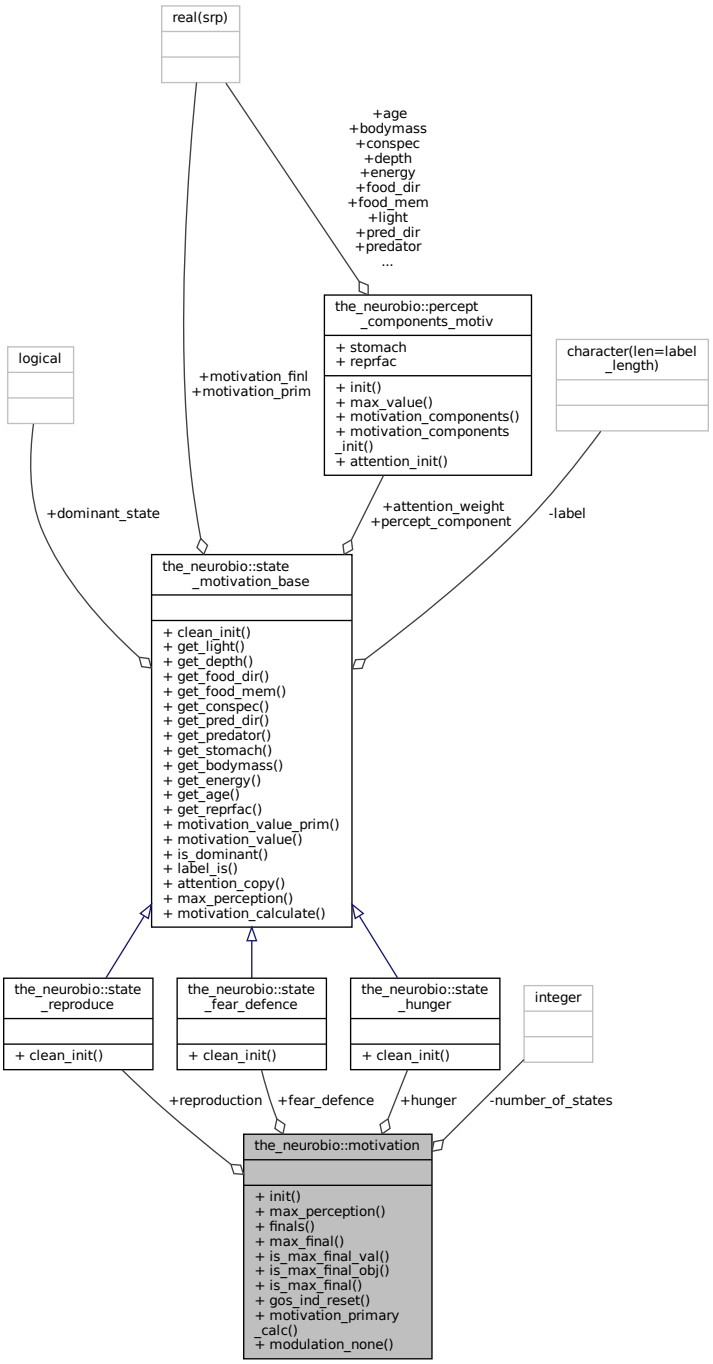
Definition at line 5648 of file m_common.f90.
The documentation for this interface was generated from the following file:

- m_common.f90

9.49 the_neurobio::motivation Type Reference

Motivation is a collection of all internal motivational states of the agent. This type is also used in defining *Expectancies* of motivations.

Collaboration diagram for the_neurobio::motivation:



Public Member Functions

- procedure, public init => motivation_init_all_zero

  Init the expectancy components to a zero state. See `the_neurobio::motivation_init_all_zero()`.
- procedure, public max_perception => motivation_max_perception_calc

  Calculate maximum value of the perception components across all motivations See `the_neurobio::motivation_max_percept`
- procedure, public finals => motivation_return_final_as_vector

  Return the vector of final motivation values for all motivational state components. See `the_neurobio::motivation_return_fi`

- procedure, public max_final => motivation_maximum_value_motivation_finl

  Calculate the maximum value of the final motivations across all motivational state components. See `the_neurobio::motivation_maximum_value_motivation_finl()`.
- procedure, public is_max_final_val => motivation_val_is_maximum_value_motivation_finl

  *Checks if the test value is the maximum **final** motivation value across all motivational state components. See* `the_neurobio::motivation_val_is_maximum_value_motivation_finl()`.
- procedure, public is_max_final_obj => motivation_val_is_maximum_value_motivation_finl_o

  *Checks if the test value is the maximum **final** motivation value across all motivational state components. See* `the_neurobio::motivation_val_is_maximum_value_motivation_finl_o()`.
- generic, public is_max_final => is_max_final_val, is_max_final_obj
- procedure, public gos_ind_reset => motivation_reset_gos_indicators

  Reset all GOS indicators for this motivation object. See `the_neurobio::motivation_reset_gos_indicators()`.
- procedure, public motivation_primary_calc => motivation_primary_sum_components

  *Functions calculating the overall **motivation state values**. @important These functions calculate the **motivation state** of the agent. This is a kind of a summator for the many perception-specific state components into the unitary inner **motivation state**. See* `the_neurobio::motivation_primary_sum_components()`.
- procedure, public modulation_none => motivation_modulation_absent

  *Functions re-calculating the overall motivation values after **modulation**.*

Public Attributes

- type(state_hunger) hunger
- type(state_fear_defence) fear_defence
- type(state_reproduce) reproduction

Private Attributes

- integer, private number_of_states

9.49.1 Detailed Description

Motivation is a collection of all internal motivational states of the agent. This type is also used in defining *Expectancies* of motivations.
Definition at line 1101 of file m_neuro.f90.

9.49.2 Member Function/Subroutine Documentation

9.49.2.1 init()

`procedure, public the_neurobio::motivation::init`
Init the expectancy components to a zero state. See `the_neurobio::motivation_init_all_zero()`.
Definition at line 1116 of file m_neuro.f90.

9.49.2.2 max_perception()

`procedure, public the_neurobio::motivation::max_perception`
Calculate maximum value of the perception components across all motivations See `the_neurobio::motivation_max_perce`
Definition at line 1120 of file m_neuro.f90.

9.49.2.3 finals()

procedure, public the_neurobio::motivation::finals
Return the vector of final motivation values for all motivational state components. See the_neurobio::motivation_return_
Definition at line 1124 of file m_neuro.f90.

9.49.2.4 max_final()

procedure, public the_neurobio::motivation::max_final
Calculate the maximum value of the final motivations across all motivational state components. See the_neurobio::motivation_maximum_value_motivation_finl().
Definition at line 1128 of file m_neuro.f90.

9.49.2.5 is_max_final_val()

procedure, public the_neurobio::motivation::is_max_final_val
Checks if the test value is the maximum **final** motivation value across all motivational state components. See the_neurobio::motivation_val_is_maximum_value_motivation_finl().
Definition at line 1132 of file m_neuro.f90.

9.49.2.6 is_max_final_obj()

procedure, public the_neurobio::motivation::is_max_final_obj
Checks if the test value is the maximum **final** motivation value across all motivational state components. See the_neurobio::motivation_val_is_maximum_value_motivation_finl_o().
Definition at line 1137 of file m_neuro.f90.

9.49.2.7 is_max_final()

generic, public the_neurobio::motivation::is_max_final
Definition at line 1139 of file m_neuro.f90.

9.49.2.8 gos_ind_reset()

procedure, public the_neurobio::motivation::gos_ind_reset
Reset all GOS indicators for this motivation object. See the_neurobio::motivation_reset_gos_indicators().
Definition at line 1142 of file m_neuro.f90.

9.49.2.9 motivation_primary_calc()

procedure, public the_neurobio::motivation::motivation_primary_calc
Functions calculating the overall **motivation state values**. @important These functions calculate the **motivation state** of the agent. This is a kind of a summator for the many perception-specific state components into the unitary inner **motivation state**. See the_neurobio::motivation_primary_sum_components().
Definition at line 1149 of file m_neuro.f90.

9.49.2.10 modulation_none()

procedure, public the_neurobio::motivation::modulation_none
Functions re-calculating the overall motivation values after **modulation**.

Note

Modulation modifies the motivation value based on other properties of the agent with effect coefficients depending on the genome. See `the_neurobio::motivation_modulation_absent()`.

Definition at line 1157 of file m_neuro.f90.

9.49.3 Member Data Documentation

9.49.3.1 hunger

`type(state_hunger) the_neurobio::motivation::hunger`

- **hunger** is the state of the_neurobio::state_hunger.

Definition at line 1103 of file m_neuro.f90.

9.49.3.2 fear_defence

`type(state_fear_defence) the_neurobio::motivation::fear_defence`

- **fear state** is the state of the_neurobio::state_fear_defence;

Definition at line 1105 of file m_neuro.f90.

9.49.3.3 reproduction

`type(state_reproduce) the_neurobio::motivation::reproduction`

- **reproduction** is the state of the_neurobio::state_reproduce;

Definition at line 1107 of file m_neuro.f90.

9.49.3.4 number_of_states

`integer, private the_neurobio::motivation::number_of_states [private]`

- `number_of_states` is a private value indicating the total number of motivational states. It is initialised to 4 in the the_neurobio::motivation_init_all_zero().

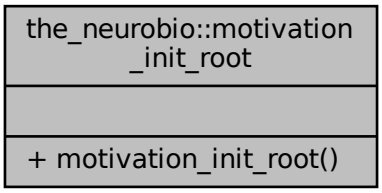
Definition at line 1112 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

9.50 the_neurobio::motivation_init_root Interface Reference

Abstract interface for the deferred **init** function `clean_init` that has to be overridden by each object that extends the basic motivational state type.

Collaboration diagram for the_neurobio::motivation_init_root:



Public Member Functions

- elemental subroutine motivation_init_root (this)

9.50.1 Detailed Description

Abstract interface for the deferred **init** function `clean_init` that has to be overridden by each object that extends the basic motivational state type.
Definition at line 1058 of file m_neuro.f90.

9.50.2 Constructor & Destructor Documentation

9.50.2.1 motivation_init_root()

```
elemental subroutine the_neurobio::motivation_init_root::motivation_init_root (
            class(state_motivation_base), intent(inout) this )
```

Parameters

in,out	*this*	

Warning

Import base type. Without import gfortran issues this error: 'Error: Derived type 'state_motivation_base' at (1) is being used before it is defined`.

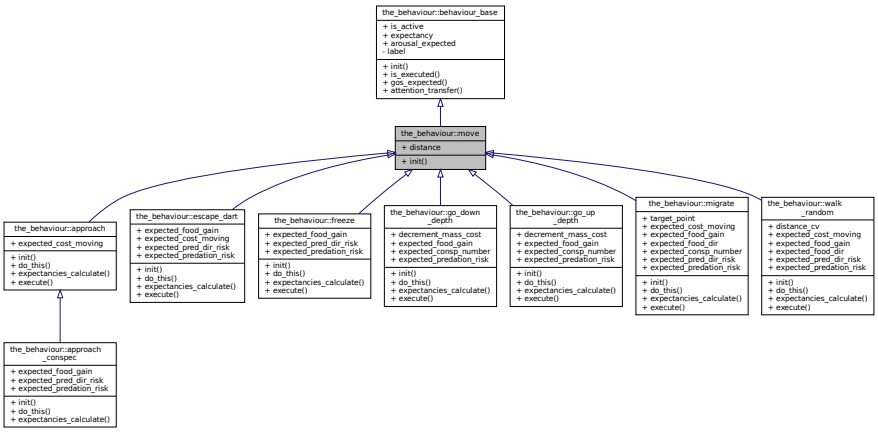
Definition at line 1058 of file m_neuro.f90.
The documentation for this interface was generated from the following file:

- m_neuro.f90

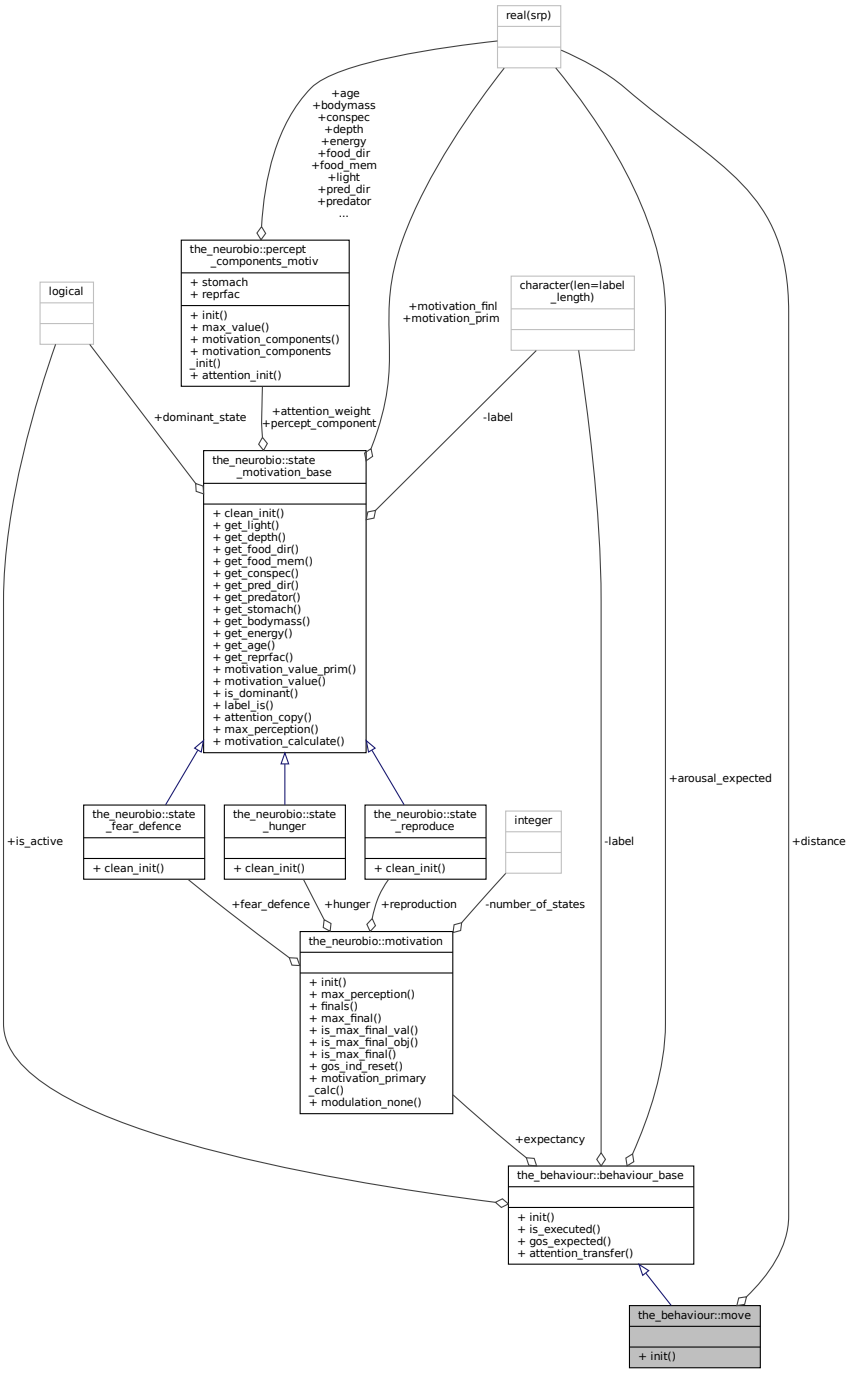
9.51 the_behaviour::move Type Reference

Movement is an umbrella abstract type linked with spatial movement.

Inheritance diagram for the_behaviour::move:



Collaboration diagram for the_behaviour::move:



Public Member Functions

- procedure(move_init_root), deferred, public init

  The the_behaviour::move::init() is a deferred function that is overriden by each extension object `init` method.

Public Attributes

- real(srp) distance

Movement is described by its absolute distance.

9.51.1 Detailed Description

Movement is an umbrella abstract type linked with spatial movement.
Definition at line 83 of file m_behav.f90.

9.51.2 Member Function/Subroutine Documentation

9.51.2.1 init()

`procedure(move_init_root), deferred, public the_behaviour::move::init`
The the_behaviour::move::init() is a deferred function that is overriden by each extension object `init` method.
Definition at line 94 of file m_behav.f90.

9.51.3 Member Data Documentation

9.51.3.1 distance

`real(srp) the_behaviour::move::distance`
Movement is described by its absolute distance.

Note

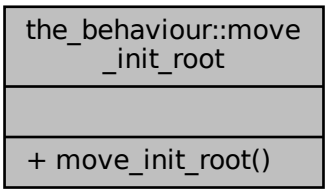
Note that the expected cost of movement is implemented separately in each derived class because calculations of the movement cost are specific to each (derived) behavioural component (e.g. the_behaviour::freeze and the_behaviour::go_down_depth.

Definition at line 90 of file m_behav.f90.
The documentation for this type was generated from the following file:

- m_behav.f90

9.52 the_behaviour::move_init_root Interface Reference

Abstract interface for the deferred **init** function that has to be overriden by each object that extends the basic behavioural component class.
Collaboration diagram for the_behaviour::move_init_root:



Public Member Functions

- elemental subroutine move_init_root (this)

9.52.1 Detailed Description

Abstract interface for the deferred **init** function that has to be overriden by each object that extends the basic behavioural component class.
Definition at line 101 of file m_behav.f90.

9.52.2 Constructor & Destructor Documentation

9.52.2.1 move_init_root()

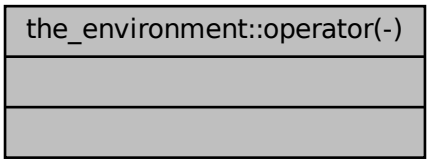
```
elemental subroutine the_behaviour::move_init_root::move_init_root (
            class(move), intent(inout) this )
```

Definition at line 101 of file m_behav.f90.
The documentation for this interface was generated from the following file:

- m_behav.f90

9.53 the_environment::operator(-) Interface Reference

Interface operator "-" for the the_environment::environment spatial container objects. Return an environment object that is shrunk by a fixed value in the 2D XxY plane. See `the_environment::environment_shrink_xy_fixed()`.
The operator can be used as follows:
Collaboration diagram for the_environment::operator(-):



9.53.1 Detailed Description

Interface operator "-" for the the_environment::environment spatial container objects. Return an environment object that is shrunk by a fixed value in the 2D XxY plane. See `the_environment::environment_shrink_xy_fixed()`.
The operator can be used as follows:

```
temp_hab = habitat_safe - 0.5_srp
```

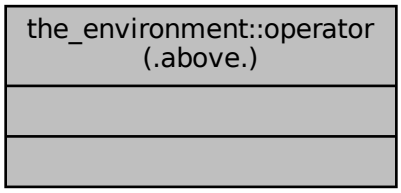
Definition at line 914 of file m_env.f90.
The documentation for this interface was generated from the following file:

- m_env.f90

9.54 the_environment::operator(.above.) Interface Reference

Interface operators .above. for spatial objects. Usage:

Collaboration diagram for the_environment::operator(.above.):



9.54.1 Detailed Description

Interface operators .above. for spatial objects. Usage:
`object1 .above. object2`
Tests the condition of `object1` is above `object2` The operator can be used in two ways:

- as an expression, with both scalar and array values:
  `parents%ind(i) .above. parents%ind(i)%perceive_food%foods_seen`
- in if blocks, only **scalars**:
  `if ( parents%ind(i) .above. parents%ind(i)%perceive_food%foods_seen(1) )`

Note

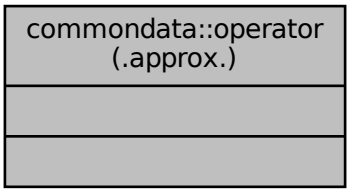
Note that the operator `.above.` refers to the "below" procedure `the_environment::spatial_check_located_bel` as the dummy parameters have reverse order in this implementation procedure.

Definition at line 880 of file m_env.f90.
The documentation for this interface was generated from the following file:

- m_env.f90

9.55 commondata::operator(.approx.) Interface Reference

"Approximatel equality" operator: Check if two real values are *approximately equal* using the commondata::is_near_zero() function. Thus function can be used for comparing two real values like the below.
Collaboration diagram for commondata::operator(.approx.):



9.55.1 Detailed Description

"Approximatel equality" operator: Check if two real values are *approximately equal* using the commondata::is_near_zero() function. Thus function can be used for comparing two real values like the below.

- The exact real comparison (incorrect due to possible rounding):
```
if ( a == b ) ...
```
- should be substituted by such *approximately equal* comparison:
```
if ( a .approx. b) ...
```

See the backend procedures commondata::float_approx_srp_operator() and commondata::float_approx_hrp_operator() for details and Float point for an introduction.

Note

This `.approx.` operator differs from the `.feq.` by a very high *epsilon* tolerance, that is, a much higher error (commondata::zero $*$ 1000.0) is tolerated.

Definition at line 5453 of file m_common.f90.
The documentation for this interface was generated from the following file:

- m_common.f90

9.56 the_environment::operator(.below.) Interface Reference

Interface operators .below. for spatial objects. Usage:
Collaboration diagram for the_environment::operator(.below.):

9.56.1 Detailed Description

Interface operators .below. for spatial objects. Usage:
```
object1 .below. object2
```
Tests the condition of `object1` is below `object2` The operator can be used in two ways:

- as an expression, with both scalar and array values:
```
parents%ind(i) .below. parents%ind(i)%perceive_food%foods_seen
```
- in if blocks, only **scalars**:
```
if ( parents%ind(i) .below. parents%ind(i)%perceive_food%foods_seen(1) )
```

Note

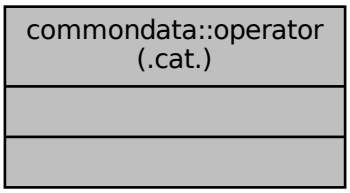
Note that the operator `.below.` refers to the "above" procedure `the_environment::spatial_check_located_abo` as the dummy parameters have reverse order in this implementation procedure.

Definition at line 902 of file m_env.f90.
The documentation for this interface was generated from the following file:

- m_env.f90

9.57 commondata::operator(.cat.) Interface Reference

Concatenate two arrays a and b. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.
Collaboration diagram for commondata::operator(.cat.):



9.57.1 Detailed Description

Concatenate two arrays a and b. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.

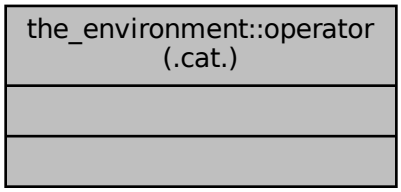
```
a .cat. b ! equivalent to [a, b]
```

See commondata::stack2arrays_r() and commondata::stack2arrays_i() for details.
Definition at line 5483 of file m_common.f90.
The documentation for this interface was generated from the following file:

- m_common.f90

9.58 the_environment::operator(.cat.) Interface Reference

Interface operator to concatenate two arrays of the spatial the_environment::spatial or spatial moving the_environment::spatial_moving objects.
Collaboration diagram for the_environment::operator(.cat.):



9.58.1 Detailed Description

Interface operator to concatenate two arrays of the spatial the_environment::spatial or spatial moving the_environment::spatial_moving objects.

```
object1%location() .cat. object2%location()
```

See the_environment::spatial_stack2arrays() and the_environment::spatial_moving_stack2arrays() for backend implementation.

Warning

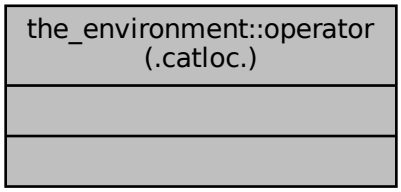
This operator works with fixed **types** rather than class. All input and output parameters are defined as **type**, so this is not class-safe.

Definition at line 821 of file m_env.f90.
The documentation for this interface was generated from the following file:

- m_env.f90

9.59 the_environment::operator(.catloc.) Interface Reference

Interface operator to concatenate the **location** components of two arrays ofthe_environment::spatial **class** objects.
Collaboration diagram for the_environment::operator(.catloc.):



9.59.1 Detailed Description

Interface operator to concatenate the **location** components of two arrays ofthe_environment::spatial **class** objects.
`all_objects%position%( object1 .catloc. object2 )`

Note

Unlike the .cat. operator implemented using the the_environment::spatial_stack2arrays() and the_environment::spatial_moving_s methods, this procedure is class-safe and can be used with any class upwards, but it concatenates **only** the location data (returns **type** the_environment::spatial).

Definition at line 837 of file m_env.f90.
The documentation for this interface was generated from the following file:

- m_env.f90

9.60 the_environment::operator(.contains.) Interface Reference

Interface operators `.contains.` for testing whether an environment object (first argument) contains a `SPATIAL` object (second argument). Usage:

Collaboration diagram for the_environment::operator(.contains.):

9.60.1 Detailed Description

Interface operators `.contains.` for testing whether an environment object (first argument) contains a `SPATIAL` object (second argument). Usage:

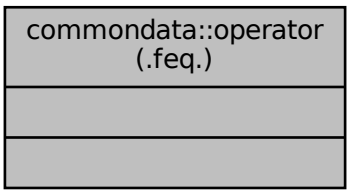
```
if ( environment .contains. object ) then
```

See `the_environment::environment_check_located_within_3d()`.
Definition at line 858 of file m_env.f90.
The documentation for this interface was generated from the following file:

- m_env.f90

9.61 commondata::operator(.feq.) Interface Reference

"Float equality" operator: Check if two real values are *nearly equal* using the commondata::is_near_zero() function. Thus function can be used for comparing two real values like the below.
Collaboration diagram for commondata::operator(.feq.):



9.61.1 Detailed Description

"Float equality" operator: Check if two real values are *nearly equal* using the commondata::is_near_zero() function. Thus function can be used for comparing two real values like the below.

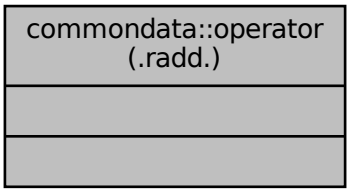
- The exact real comparison (incorrect due to possible rounding):
  ```
  if ( a == b ) ...
  ```
- should be substituted by such comparison:
  ```
  if ( a .feq. b) ...
  ```

See the backend procedures commondata::float_equal_srp_operator() and commondata::float_equal_hrp_operator() for details and Float point for an introduction.
Definition at line 5428 of file m_common.f90.
The documentation for this interface was generated from the following file:

- m_common.f90

9.62 commondata::operator(.radd.) Interface Reference

Interface operator `.radd.` performs a random addition or subtraction of two numbers with equal probability. See `commondata::random_add_subtract()`. The operator can be used as follows:
Collaboration diagram for commondata::operator(.radd.):



9.62.1 Detailed Description

Interface operator `.radd.` performs a random addition or subtraction of two numbers with equal probability. See `commondata::random_add_subtract()`. The operator can be used as follows:

```
temp_hab = a .radd. b
```

Note

Used in the correlated random Gaussian walk routines the_environment::corwalk()

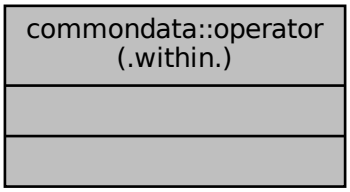
Definition at line 5523 of file m_common.f90.
The documentation for this interface was generated from the following file:

- m_common.f90

9.63 commondata::operator(.within.) Interface Reference

Interface operators `.within.` for testing whether a value (first argument) lies within the limits set by a two-element array (second argument). All the values/parameters are Fortran intrinsic types, real or integer. Usage of the operator:

Collaboration diagram for commondata::operator(.within.):



9.63.1 Detailed Description

Interface operators `.within.` for testing whether a value (first argument) lies within the limits set by a two-element array (second argument). All the values/parameters are Fortran intrinsic types, real or integer. Usage of the operator:

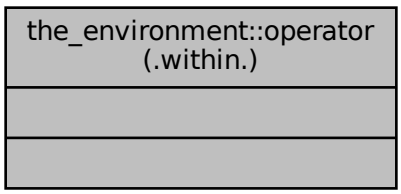
```
if ( value .within. [lower, upper] ) then
```

See `commondata::is_within_operator_r()` and `commondata::is_within_operator_i()` for more details.
Definition at line 5469 of file m_common.f90.
The documentation for this interface was generated from the following file:

- m_common.f90

9.64 the_environment::operator(.within.) Interface Reference

Interface operators `.within.` for testing whether a spatial object (first argument lies within an environment (second argument). Usage:
Collaboration diagram for the_environment::operator(.within.):



9.64.1 Detailed Description

Interface operators `.within.` for testing whether a spatial object (first argument lies within an environment (second argument). Usage:

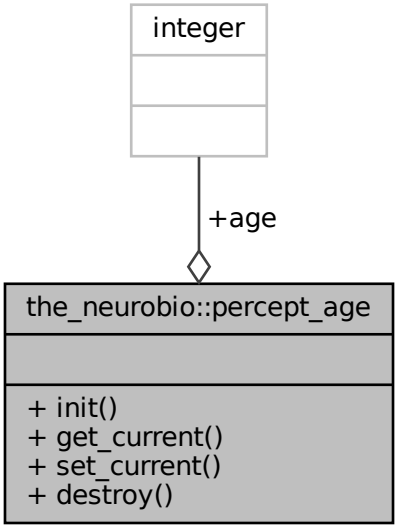
```
if ( object .within. environment ) then
```

See `the_environment::spatial_check_located_within_3d()`.
Definition at line 847 of file m_env.f90.
The documentation for this interface was generated from the following file:

- m_env.f90

9.65 the_neurobio::percept_age Type Reference

This type defines how the agent perceives its own age in terms of the model discrete time step.
Collaboration diagram for the_neurobio::percept_age:



Public Member Functions

- procedure, public init => percept_age_create_init

  *Initiate an empty **age** perception object. See `the_neurobio::percept_age_create_init()`.*
- procedure, public get_current => percept_age_get_current

  *Get the current value of the **age** reserves. See `the_neurobio::percept_age_get_current()`.*
- procedure, public set_current =>percept_age_update_current

  *Set and update the current **age** perception value. See `the_neurobio::percept_age_update_current()`.*
- procedure, public destroy => percept_age_destroy_deallocate

  *Destroy the **age** perception object and deallocate. See `the_neurobio::percept_age_destroy_deallocate()`.*

Public Attributes

- integer age

9.65.1 Detailed Description

This type defines how the agent perceives its own age in terms of the model discrete time step.
Definition at line 355 of file m_neuro.f90.

9.65.2 Member Function/Subroutine Documentation

9.65.2.1 init()

`procedure, public the_neurobio::percept_age::init`
Initiate an empty **age** perception object. See `the_neurobio::percept_age_create_init()`.
Definition at line 360 of file m_neuro.f90.

9.65.2.2 get_current()

`procedure, public the_neurobio::percept_age::get_current`
Get the current value of the **age** reserves. See `the_neurobio::percept_age_get_current()`.
Definition at line 363 of file m_neuro.f90.

9.65.2.3 set_current()

`procedure, public the_neurobio::percept_age::set_current`
Set and update the current **age** perception value. See `the_neurobio::percept_age_update_current()`.
Definition at line 366 of file m_neuro.f90.

9.65.2.4 destroy()

`procedure, public the_neurobio::percept_age::destroy`
Destroy the **age** perception object and deallocate. See `the_neurobio::percept_age_destroy_deallocate()`.
Definition at line 369 of file m_neuro.f90.

9.65.3 Member Data Documentation

9.65.3.1 age

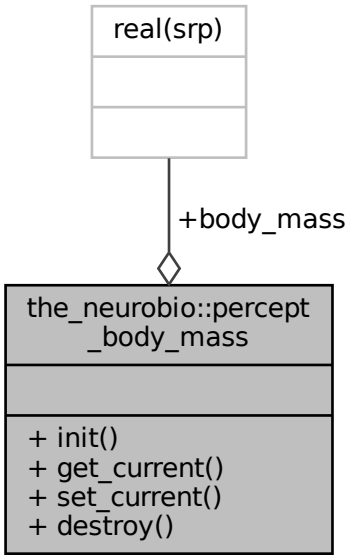
`integer the_neurobio::percept_age::age`
Definition at line 356 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

9.66 the_neurobio::percept_body_mass Type Reference

This type defines how the agent perceives its own body mass it can be important for state-dependency.

Collaboration diagram for the_neurobio::percept_body_mass:



Public Member Functions

- procedure, public init => percept_bodymass_create_init

  *Initiate an empty **body mass** perception object. See the_neurobio::percept_bodymass_create_init().*
- procedure, public get_current => percept_bodymass_get_current

  *Get the current value of the **body mass** perception. See the_neurobio::percept_bodymass_get_current().*
- procedure, public set_current => percept_bodymass_update_current

  *Set and update the current **body mass** perception value. See the_neurobio::percept_bodymass_update_current().*
- procedure, public destroy => percept_bodymass_destroy_deallocate

  *Destroy the **body mass** perception object and deallocate. See the_neurobio::percept_bodymass_destroy_deallocate*

Public Attributes

- real(srp) body_mass

  The current body mass of the agent.

9.66.1 Detailed Description

This type defines how the agent perceives its own body mass it can be important for state-dependency.
Definition at line 315 of file m_neuro.f90.

9.66.2 Member Function/Subroutine Documentation

9.66.2.1 init()

`procedure, public the_neurobio::percept_body_mass::init`

Initiate an empty **body mass** perception object. See the_neurobio::percept_bodymass_create_init().

Definition at line 321 of file m_neuro.f90.

9.66.2.2 get_current()

`procedure, public the_neurobio::percept_body_mass::get_current`
Get the current value of the **body mass** perception. See `the_neurobio::percept_bodymass_get_current()`.
Definition at line 324 of file m_neuro.f90.

9.66.2.3 set_current()

`procedure, public the_neurobio::percept_body_mass::set_current`
Set and update the current **body mass** perception value. See `the_neurobio::percept_bodymass_update_current()`.
Definition at line 327 of file m_neuro.f90.

9.66.2.4 destroy()

`procedure, public the_neurobio::percept_body_mass::destroy`
Destroy the **body mass** perception object and deallocate. See `the_neurobio::percept_bodymass_destroy_dealloca`
Definition at line 330 of file m_neuro.f90.

9.66.3 Member Data Documentation

9.66.3.1 body_mass

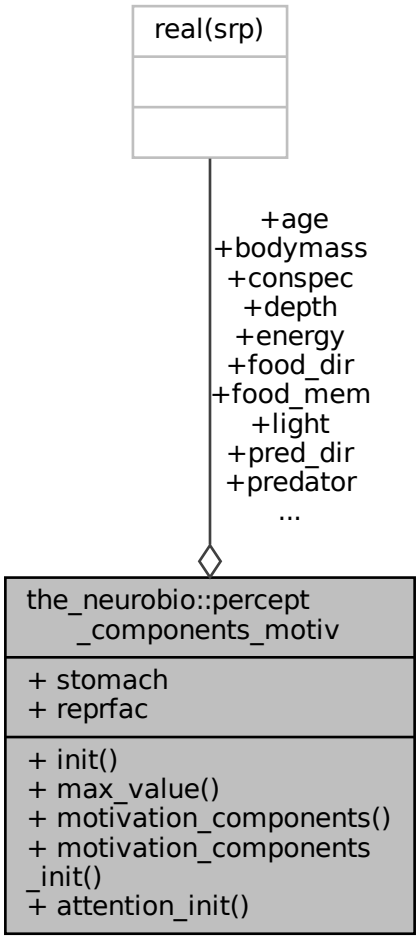
`real(srp) the_neurobio::percept_body_mass::body_mass`
The current body mass of the agent.
Definition at line 317 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

9.67 the_neurobio::percept_components_motiv Type Reference

Perceptual components of motivational states. Plugged into all `STATE_`, attention etc. These components are linked to specific inner or outer perception objects (stimuli). Their sum result(s) in the overall value of the motivation component.

Collaboration diagram for the_neurobio::percept_components_motiv:



Public Member Functions

- procedure, public init => perception_component_motivation_init_zero

  Initialise perception components for a motivation state object. See `the_neurobio::perception_component_motivation_i`

- procedure, public max_value => perception_component_maxval

  Calculate the ***maximum*** *value over all the perceptual components. See* `the_neurobio::perception_component_maxval()`.

- procedure, public motivation_components => perception_components_neuronal_response_calculate

  Calculate individual perceptual components for ***this*** *motivational state using the* ***neuronal response*** *function, for an agent. This agent has intent[in], so is* ***unchanged*** *in this procedure. Also* `motivation_components` *can take optional arbitrary (fake) perception values.*

- procedure, public motivation_components_init => perception_components_neuronal_response_init_set

  Calculate individual perceptual components for ***this*** *motivational state using the* ***neuronal response*** *function, for an agent. This agent has intent[inout], so* ***is changed*** *(gene labels reset).*

- procedure, public attention_init => perception_components_attention_weights_init

  Initialise the attention components of the emotional state to their default parameter values. Attention sets weights to individual perceptual components when the overall weighted sum is calculated. The default weights are parameters defined in `COMMONDATA`. *See* `the_neurobio::perception_components_attention_weights_init()`.

Public Attributes

- real(srp) light

  Light perception, direct environmental.
- real(srp) depth

  Depth perception, direct environmental.
- real(srp) food_dir

  Perception of directly seen food items, spatial.
- real(srp) food_mem

  Perception of the food items in the memory stack.
- real(srp) conspec

  Perception of conspecifics, spatial.
- real(srp) pred_dir

  Direct perception of predators, spatial. Based on the distance to the nearest predator.
- real(srp) predator

  General perception of predation risk, spatial. Based on a sum of the number of predators in the perception object weighted by the number of predators in the memory stack.
- real(srp) stomach

  Perception of the stomach contents, direct, internal.
- real(srp) bodymass

  Perception of the body mass, direct, internal.
- real(srp) energy

  Perception of the energy reserves, direct, internal.
- real(srp) age

  Age perception, direct internal.
- real(srp) reprfac

  Perception of the reproductive factor, based on the sex steroid hormones, calculated differently in males and females.

9.67.1 Detailed Description

Perceptual components of motivational states. Plugged into all `STATE_`, attention etc. These components are linked to specific inner or outer perception objects (stimuli). Their sum result(s) in the overall value of the motivation component.
Definition at line 875 of file m_neuro.f90.

9.67.2 Member Function/Subroutine Documentation

9.67.2.1 init()

`procedure, public the_neurobio::percept_components_motiv::init`
Initialise perception components for a motivation state object. See `the_neurobio::perception_component_motivation`
Definition at line 907 of file m_neuro.f90.

9.67.2.2 max_value()

`procedure, public the_neurobio::percept_components_motiv::max_value`
Calculate the **maximum** value over all the perceptual components. See `the_neurobio::perception_component_maxval`
Definition at line 910 of file m_neuro.f90.

9.67.2.3 motivation_components()

`procedure, public the_neurobio::percept_components_motiv::motivation_components`
Calculate individual perceptual components for **this** motivational state using the **neuronal response** function, for an agent. This agent has intent[in], so is **unchanged** in this procedure. Also `motivation_components` can take optional arbitrary (fake) perception values.

Note

This procedure is used for normal calculations of motivation components. A similar method with the agent intent[inout] the_neurobio::percept_components_motiv::motivation_components_init() is used to initialise an agent. See `the_neurobio::perception_components_neuronal_response_calculate()`.

Definition at line 922 of file m_neuro.f90.

9.67.2.4 motivation_components_init()

`procedure, public the_neurobio::percept_components_motiv::motivation_components_init`
Calculate individual perceptual components for **this** motivational state using the **neuronal response** function, for an agent. This agent has intent[inout], so **is changed** (gene labels reset).

Warning

This procedure is used only for initialisation of an agent. For normal calculation of the motivational components use the_neurobio::percept_components_motiv::motivation_components() procedure that does not change the actor agent (intent[in]). See `the_neurobio::perception_components_neuronal_response_init_set()`.

Definition at line 932 of file m_neuro.f90.

9.67.2.5 attention_init()

`procedure, public the_neurobio::percept_components_motiv::attention_init`
Initialise the attention components of the emotional state to their default parameter values. Attention sets weights to individual perceptual components when the overall weighted sum is calculated. The default weights are parameters defined in `COMMONDATA`. See `the_neurobio::perception_components_attention_weights_init()`.
Definition at line 939 of file m_neuro.f90.

9.67.3 Member Data Documentation

9.67.3.1 light

`real(srp) the_neurobio::percept_components_motiv::light`
Light perception, direct environmental.
Definition at line 877 of file m_neuro.f90.

9.67.3.2 depth

`real(srp) the_neurobio::percept_components_motiv::depth`
Depth perception, direct environmental.
Definition at line 879 of file m_neuro.f90.

9.67.3.3 food_dir

`real(srp) the_neurobio::percept_components_motiv::food_dir`
Perception of directly seen food items, spatial.
Definition at line 881 of file m_neuro.f90.

9.67.3.4 food_mem

real(srp) the_neurobio::percept_components_motiv::food_mem
Perception of the food items in the memory stack.
Definition at line 883 of file m_neuro.f90.

9.67.3.5 conspec

real(srp) the_neurobio::percept_components_motiv::conspec
Perception of conspecifics, spatial.
Definition at line 885 of file m_neuro.f90.

9.67.3.6 pred_dir

real(srp) the_neurobio::percept_components_motiv::pred_dir
Direct perception of predators, spatial. Based on the distance to the nearest predator.
Definition at line 888 of file m_neuro.f90.

9.67.3.7 predator

real(srp) the_neurobio::percept_components_motiv::predator
General perception of predation risk, spatial. Based on a sum of the number of predators in the perception object weighted by the number of predators in the memory stack.
Definition at line 892 of file m_neuro.f90.

9.67.3.8 stomach

real(srp) the_neurobio::percept_components_motiv::stomach
Perception of the stomach contents, direct, internal.
Definition at line 894 of file m_neuro.f90.

9.67.3.9 bodymass

real(srp) the_neurobio::percept_components_motiv::bodymass
Perception of the body mass, direct, internal.
Definition at line 896 of file m_neuro.f90.

9.67.3.10 energy

real(srp) the_neurobio::percept_components_motiv::energy
Perception of the energy reserves, direct, internal.
Definition at line 898 of file m_neuro.f90.

9.67.3.11 age

real(srp) the_neurobio::percept_components_motiv::age
Age perception, direct internal.
Definition at line 900 of file m_neuro.f90.

9.67.3.12 reprfac

`real(srp) the_neurobio::percept_components_motiv::reprfac`

Perception of the reproductive factor, based on the sex steroid hormones, calculated differently in males and females.

Definition at line 903 of file m_neuro.f90.

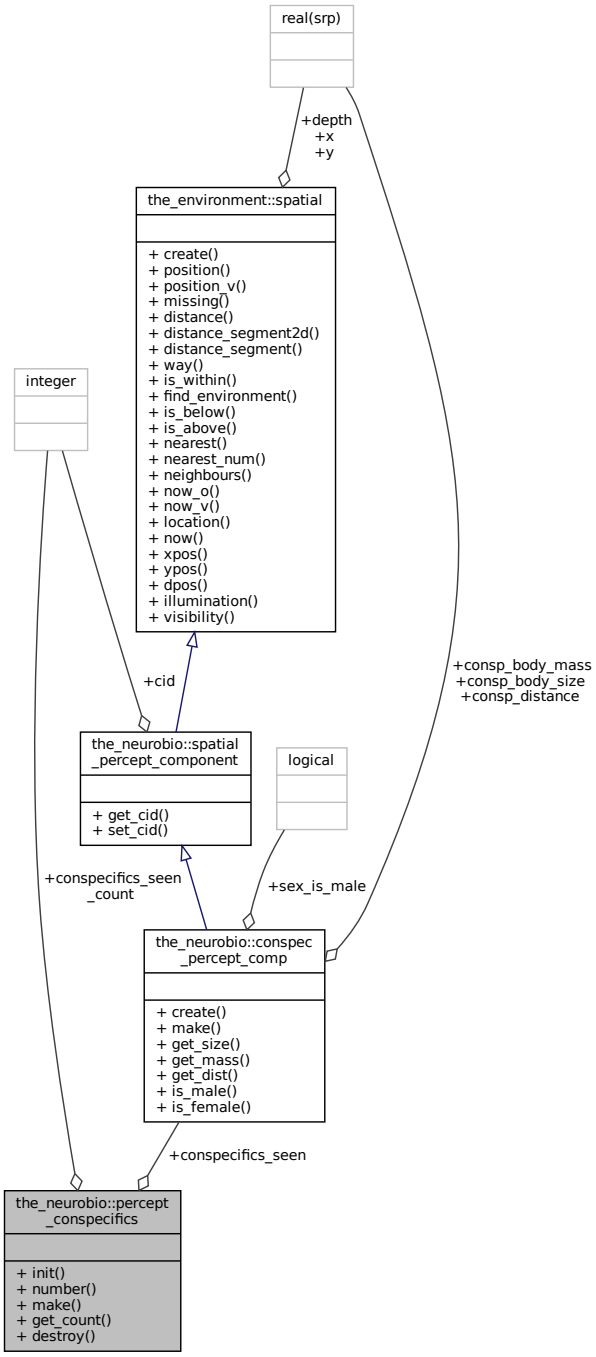
The documentation for this type was generated from the following file:

- m_neuro.f90

9.68 the_neurobio::percept_conspecifics Type Reference

This type defines how the agent perceives conspecifics.

Collaboration diagram for the_neurobio::percept_conspecifics:



Public Member Functions

- procedure, public init => percept_consp_create_init

  Create conspecifics perception object, it is an array of conspecific perception components. See `the_neurobio::percept_consp_`

- procedure, public number => percept_consp_number_seen

  Set the total number of conspecifics perceived (seen) in the conspecific perception object. See `the_neurobio::percept_consp_n`

- procedure, public make => percept_consp_make_fill_arrays

  Make the conspecifics perception object, fill it with the actual arrays. See `the_neurobio::percept_consp_make_fill_array`

- procedure, public get_count => percept_consp_get_count_seen

  Get the number (count) of conspecifics seen. See the_neurobio::percept_consp_get_count_seen().
- procedure, public destroy => percept_consp_destroy_deallocate

  Deallocate and delete a conspecific perception object. See the_neurobio::percept_consp_destroy_deallocate().

Public Attributes

- type(conspec_percept_comp), dimension(:), allocatable conspecifics_seen

  An array of conspecifics seen in proximity, within the visual range.
- integer conspecifics_seen_count

  The number of conspecifics seen.

9.68.1 Detailed Description

This type defines how the agent perceives conspecifics.
Definition at line 183 of file m_neuro.f90.

9.68.2 Member Function/Subroutine Documentation

9.68.2.1 init()

`procedure, public the_neurobio::percept_conspecifics::init`
Create conspecifics perception object, it is an array of conspecific perception components. See the_neurobio::percept_cons
Definition at line 195 of file m_neuro.f90.

9.68.2.2 number()

`procedure, public the_neurobio::percept_conspecifics::number`
Set the total number of conspecifics perceived (seen) in the conspecific perception object. See the_neurobio::percept_cons
Definition at line 199 of file m_neuro.f90.

9.68.2.3 make()

`procedure, public the_neurobio::percept_conspecifics::make`
Make the conspecifics perception object, fill it with the actual arrays. See the_neurobio::percept_consp_make_fill_ar
Definition at line 203 of file m_neuro.f90.

9.68.2.4 get_count()

`procedure, public the_neurobio::percept_conspecifics::get_count`
Get the number (count) of conspecifics seen. See the_neurobio::percept_consp_get_count_seen().
Definition at line 206 of file m_neuro.f90.

9.68.2.5 destroy()

`procedure, public the_neurobio::percept_conspecifics::destroy`
Deallocate and delete a conspecific perception object. See the_neurobio::percept_consp_destroy_deallocate().
Definition at line 209 of file m_neuro.f90.

9.68.3 Member Data Documentation

9.68.3.1 conspecifics_seen

```
type(conspec_percept_comp), dimension(:), allocatable the_neurobio::percept_conspecifics↩
::conspecifics_seen
```

An array of conspecifics seen in proximity, within the visual range.

Note

Perception of conspecifics is implemented similar to food items. Conspecific perception object the_neurobio::percept_conspecifics is a simple the_environment::spatial object.

Definition at line 188 of file m_neuro.f90.

9.68.3.2 conspecifics_seen_count

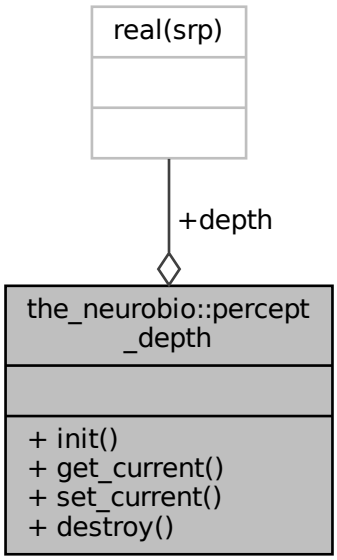
```
integer the_neurobio::percept_conspecifics::conspecifics_seen_count
```

The number of conspecifics seen.
Definition at line 190 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

9.69 the_neurobio::percept_depth Type Reference

Perception of the current depth horizon.
Collaboration diagram for the_neurobio::percept_depth:



Public Member Functions

- procedure, public init => percept_depth_create_init

  Make en empty depth perception component. See the_neurobio::percept_depth_create_init().
- procedure, public get_current => percept_depth_get_current

*Get the current perception of the **depth**. See the_neurobio::percept_depth_get_current().*

- procedure, public set_current => percept_depth_set_current

  *Set the current **depth** level into the perception component. See the_neurobio::percept_depth_set_current().*

- procedure, public destroy => percept_depth_destroy_deallocate

  *Destroy / deallocate **depth** perception component. See the_neurobio::percept_depth_destroy_deallocate().*

Public Attributes

- real(srp) depth

9.69.1 Detailed Description

Perception of the current depth horizon.
Definition at line 418 of file m_neuro.f90.

9.69.2 Member Function/Subroutine Documentation

9.69.2.1 init()

`procedure, public the_neurobio::percept_depth::init`
Make en empty depth perception component. See the_neurobio::percept_depth_create_init().
Definition at line 423 of file m_neuro.f90.

9.69.2.2 get_current()

`procedure, public the_neurobio::percept_depth::get_current`
Get the current perception of the **depth**. See the_neurobio::percept_depth_get_current().
Definition at line 426 of file m_neuro.f90.

9.69.2.3 set_current()

`procedure, public the_neurobio::percept_depth::set_current`
Set the current **depth** level into the perception component. See the_neurobio::percept_depth_set_current().
Definition at line 429 of file m_neuro.f90.

9.69.2.4 destroy()

`procedure, public the_neurobio::percept_depth::destroy`
Destroy / deallocate **depth** perception component. See the_neurobio::percept_depth_destroy_deallocate().
Definition at line 432 of file m_neuro.f90.

9.69.3 Member Data Documentation

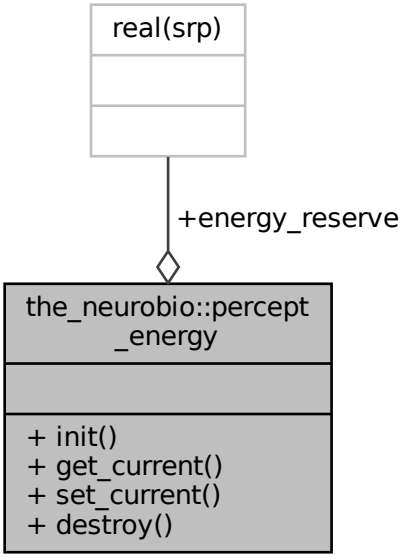
9.69.3.1 depth

`real(srp) the_neurobio::percept_depth::depth`
Definition at line 419 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

9.70 the_neurobio::percept_energy Type Reference

This type defines how the agent perceives its own energy reserves it can be important for state-dependency.
Collaboration diagram for the_neurobio::percept_energy:



Public Member Functions

- procedure, public init => percept_energy_create_init

  *Initiate an empty **energy** perception object. See `the_neurobio::percept_energy_create_init()`.*
- procedure, public get_current => percept_energy_get_current

  *Get the current value of the **energy** reserves. See `the_neurobio::percept_energy_get_current()`.*
- procedure, public set_current =>percept_energy_update_current

  *Set and update the current **energy** perception value. See `the_neurobio::percept_energy_update_current()`.*
- procedure, public destroy => percept_energy_destroy_deallocate

  *Destroy the **energy** perception object and deallocate. See `the_neurobio::percept_energy_destroy_deallocate()`.*

Public Attributes

- real(srp) energy_reserve

  The current energy reserves of the agent.

9.70.1 Detailed Description

This type defines how the agent perceives its own energy reserves it can be important for state-dependency.
Definition at line 335 of file m_neuro.f90.

9.70.2 Member Function/Subroutine Documentation

9.70.2.1 init()

`procedure, public the_neurobio::percept_energy::init`
Initiate an empty **energy** perception object. See `the_neurobio::percept_energy_create_init()`.
Definition at line 341 of file m_neuro.f90.

9.70.2.2 get_current()

`procedure, public the_neurobio::percept_energy::get_current`
Get the current value of the **energy** reserves. See `the_neurobio::percept_energy_get_current()`.
Definition at line 344 of file m_neuro.f90.

9.70.2.3 set_current()

`procedure, public the_neurobio::percept_energy::set_current`
Set and update the current **energy** perception value. See `the_neurobio::percept_energy_update_current()`.
Definition at line 347 of file m_neuro.f90.

9.70.2.4 destroy()

`procedure, public the_neurobio::percept_energy::destroy`
Destroy the **energy** perception object and deallocate. See `the_neurobio::percept_energy_destroy_deallocate()`.
Definition at line 350 of file m_neuro.f90.

9.70.3 Member Data Documentation

9.70.3.1 energy_reserve

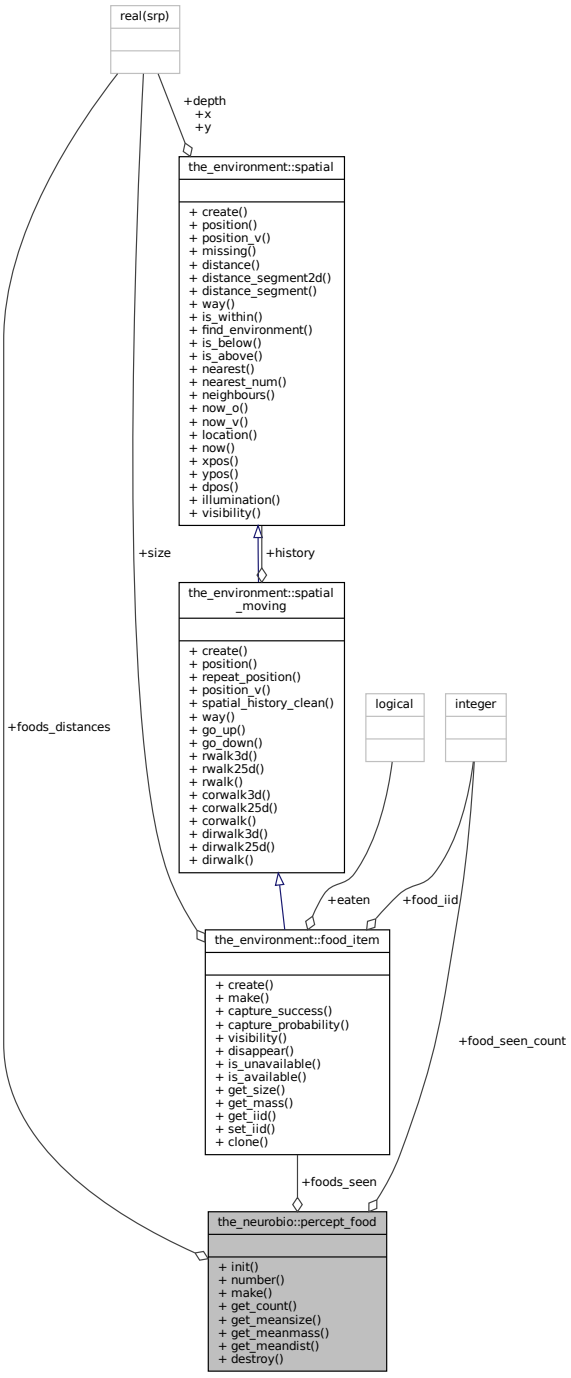
`real(srp) the_neurobio::percept_energy::energy_reserve`
The current energy reserves of the agent.
Definition at line 337 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

9.71 the_neurobio::percept_food Type Reference

This type defines how the agent perceives food items. The food perception object the_neurobio::percept_food is basically an array of food objects within the visual range of the agent plus distances to the agent. This is the "objective" perception container, reflecting the "real world". We introduce a perception error when perception object is analysed by the agent's neurobiological system.

Collaboration diagram for the_neurobio::percept_food:



Public Member Functions

- procedure, public init => percept_food_create_init

  *Initiate an empty **food** perception object with known number of components. See* `the_neurobio::percept_food_create_ini`

- procedure, public number => percept_food_number_seen
- procedure, public make => percept_food_make_fill_arrays

  Set the total number of food items perceived (seen) in the food perception object. Do not reallocate the perception object components with respect to this new number yet. See `the_neurobio::percept_food_make_fill_arrays()`.

- procedure, public get_count => percept_food_get_count_found

  Get the number (count) of food items seen. See `the_neurobio::percept_food_get_count_found()`.
- procedure, public get_meansize => percept_food_get_meansize_found

  Get the average size of food items seen. See `the_neurobio::percept_food_get_meansize_found()`.
- procedure, public get_meanmass => percept_food_get_meanmass_found

  Get the average mass of food items seen. See `the_neurobio::percept_food_get_meanmass_found()`.
- procedure, public get_meandist => percept_food_get_meandist_found

  Get the average distance tot the food items seen. See `the_neurobio::percept_food_get_meandist_found()`.
- procedure, public destroy => percept_food_destroy_deallocate

  *Deallocate and delete a **food** perception object. See* `the_neurobio::percept_food_destroy_deallocate()`.

Public Attributes

- type(food_item), dimension(:), allocatable foods_seen

  An array of food items found within the visual range, limited by the maximum order of partial indexing `commondata::food_select_items_index_partial`.
- real(srp), dimension(:), allocatable foods_distances

  An array of distances towards each of the food items.
- integer food_seen_count

  Total number of food items within the visual range of the agent. must not exceed the `commondata::food_select_items_index_` *parameter.*

9.71.1 Detailed Description

This type defines how the agent perceives food items. The food perception object the_neurobio::percept_food is basically an array of food objects within the visual range of the agent plus distances to the agent. This is the "objective" perception container, reflecting the "real world". We introduce a perception error when perception object is analysed by the agent's neurobiological system.
Definition at line 46 of file m_neuro.f90.

9.71.2 Member Function/Subroutine Documentation

9.71.2.1 init()

`procedure, public the_neurobio::percept_food::init`
Initiate an empty **food** perception object with known number of components. See `the_neurobio::percept_food_create_i`
Definition at line 87 of file m_neuro.f90.

9.71.2.2 number()

`procedure, public the_neurobio::percept_food::number`
Definition at line 89 of file m_neuro.f90.

9.71.2.3 make()

`procedure, public the_neurobio::percept_food::make`
Set the total number of food items perceived (seen) in the food perception object. Do not reallocate the perception object components with respect to this new number yet. See `the_neurobio::percept_food_make_fill_arrays()`.
Definition at line 94 of file m_neuro.f90.

9.71.2.4 get_count()

`procedure, public the_neurobio::percept_food::get_count`
Get the number (count) of food items seen. See the_neurobio::percept_food_get_count_found().
Definition at line 97 of file m_neuro.f90.

9.71.2.5 get_meansize()

`procedure, public the_neurobio::percept_food::get_meansize`
Get the average size of food items seen. See the_neurobio::percept_food_get_meansize_found().
Definition at line 100 of file m_neuro.f90.

9.71.2.6 get_meanmass()

`procedure, public the_neurobio::percept_food::get_meanmass`
Get the average mass of food items seen. See the_neurobio::percept_food_get_meanmass_found().
Definition at line 103 of file m_neuro.f90.

9.71.2.7 get_meandist()

`procedure, public the_neurobio::percept_food::get_meandist`
Get the average distance tot the food items seen. See the_neurobio::percept_food_get_meandist_found().
Definition at line 106 of file m_neuro.f90.

9.71.2.8 destroy()

`procedure, public the_neurobio::percept_food::destroy`
Deallocate and delete a **food** perception object. See the_neurobio::percept_food_destroy_deallocate().
Definition at line 109 of file m_neuro.f90.

9.71.3 Member Data Documentation

9.71.3.1 foods_seen

`type(food_item), dimension(:), allocatable the_neurobio::percept_food::foods_seen`
An array of food items found within the visual range, limited by the maximum order of partial indexing commondata::food_select_items_index_partial.

Note

> **Food perception** is quite complex to implement as it requires determining individual food items within the current visual range of the agent. There are, however, potentially thousands (or millions) of food items in the food resource, each of the food items is stochastic (e.g. they have different sizes), so visual range differ for each item and each agent should determine food items in its proximity at numerous time steps of the model. This means repeating huge loops many times for each agent at each time step. This is approached by array segmentation: the perception object is obtained by *partial indexing* of a very limited number (=commondata::food_select_items_index_partial) of only the nearest food items, the agent's visual range is then determined for each of this nearest neighbouring food items, and finally those food items that individually fall within the visual range are included into the perception object.

Definition at line 77 of file m_neuro.f90.

9.71.3.2 foods_distances

`real(srp), dimension(:), allocatable the_neurobio::percept_food::foods_distances`
An array of distances towards each of the food items.
Definition at line 79 of file m_neuro.f90.

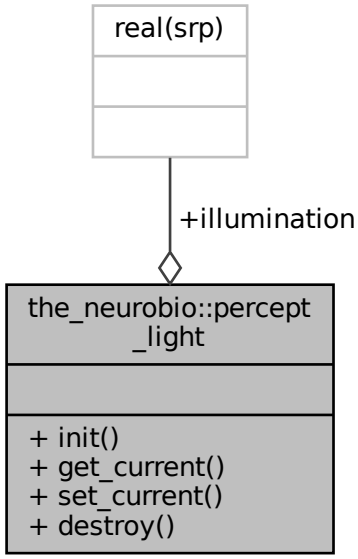
9.71.3.3 food_seen_count

`integer the_neurobio::percept_food::food_seen_count`
Total number of food items within the visual range of the agent. must not exceed the `commondata::food_select_items_inde` parameter.
Definition at line 83 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

9.72 the_neurobio::percept_light Type Reference

Perception of the ambient illumination. This is a very simple perception component, singular and static.
Collaboration diagram for the_neurobio::percept_light:



Public Member Functions

- procedure, public init => percept_light_create_init

  Make en empty light perception component. See `the_neurobio::percept_light_create_init()`.
- procedure, public get_current => percept_light_get_current

  Get the current perception of the illumination. See `the_neurobio::percept_light_get_current()`.
- procedure, public set_current => percept_light_set_current

  *Set the current **light** level into the perception component. See `the_neurobio::percept_light_set_current()`.*
- procedure, public destroy => percept_light_destroy_deallocate

  *Destroy / deallocate **light** perception component. See `the_neurobio::percept_light_destroy_deallocate()`.*

Public Attributes

- real(srp) illumination

9.72.1 Detailed Description

Perception of the ambient illumination. This is a very simple perception component, singular and static.
Definition at line 400 of file m_neuro.f90.

9.72.2 Member Function/Subroutine Documentation

9.72.2.1 init()

`procedure, public the_neurobio::percept_light::init`
Make en empty light perception component. See `the_neurobio::percept_light_create_init()`.
Definition at line 405 of file m_neuro.f90.

9.72.2.2 get_current()

`procedure, public the_neurobio::percept_light::get_current`
Get the current perception of the illumination. See `the_neurobio::percept_light_get_current()`.
Definition at line 408 of file m_neuro.f90.

9.72.2.3 set_current()

`procedure, public the_neurobio::percept_light::set_current`
Set the current **light** level into the perception component. See `the_neurobio::percept_light_set_current()`.
Definition at line 411 of file m_neuro.f90.

9.72.2.4 destroy()

`procedure, public the_neurobio::percept_light::destroy`
Destroy / deallocate **light** perception component. See `the_neurobio::percept_light_destroy_deallocate()`.
Definition at line 414 of file m_neuro.f90.

9.72.3 Member Data Documentation

9.72.3.1 illumination

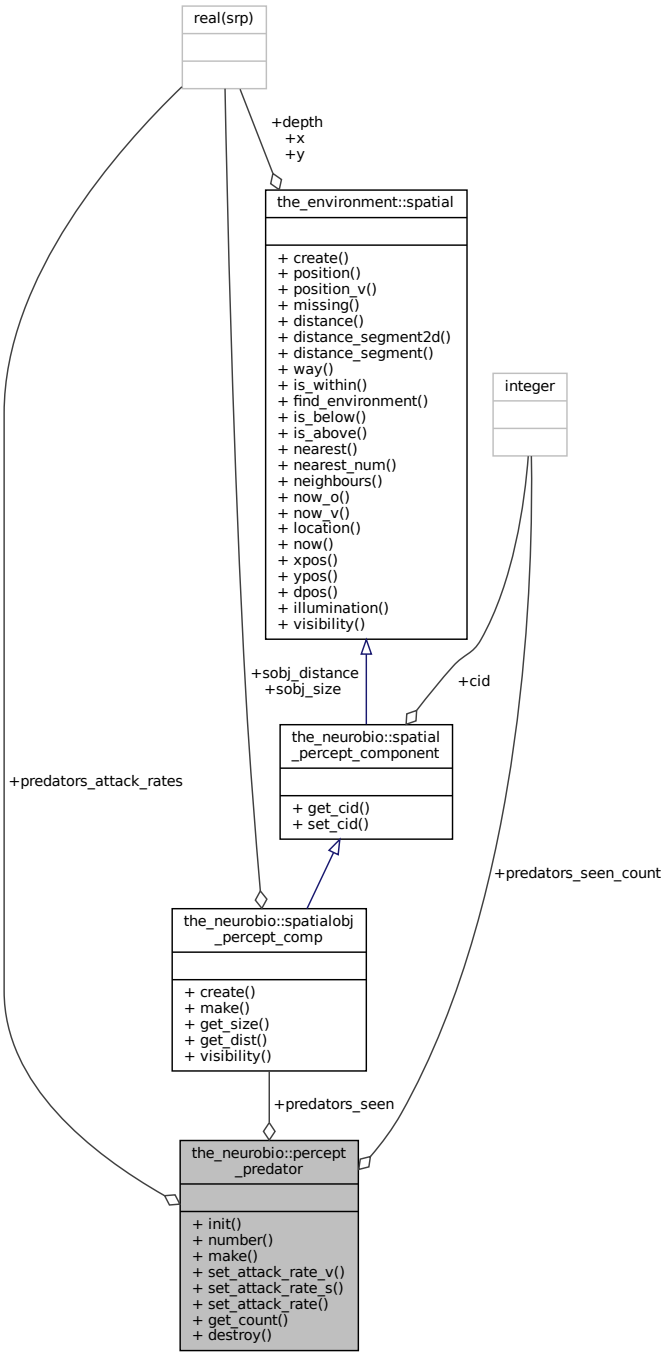
`real(srp) the_neurobio::percept_light::illumination`
Definition at line 401 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

9.73 the_neurobio::percept_predator Type Reference

This type defines how the agent perceives a predator.

Collaboration diagram for the_neurobio::percept_predator:



Public Member Functions

- procedure, public init => percept_predator_create_init

  *Create **conspecifics** perception object, it is an array of conspecific perception components. See* `the_neurobio::percept_preda`
- procedure, public number => percept_predator_number_seen

  *Set the total number of **predators** perceived (seen) in the predator perception object. See* `the_neurobio::percept_predator_n`
- procedure, public make => percept_predator_make_fill_arrays

  *Make the **predator** perception object, fill it with the actual arrays. See* `the_neurobio::percept_predator_make_fill_arra`

- procedure, public set_attack_rate_v => percept_predator_set_attack_rate_vector

  Set an array of the attack rates for the predator perception object. See the_neurobio::percept_predator_set_attack_rat
- procedure, public set_attack_rate_s => percept_predator_set_attack_rate_scalar

  Set an array of the attack rates for the predator perception object. See the_neurobio::percept_predator_set_attack_rat
- generic, public set_attack_rate => set_attack_rate_v, set_attack_rate_s

  A generic interface to set the attack rates for the predator perception object. See the_neurobio::percept_predator_set_att and the_neurobio::percept_predator_set_attack_rate_scalar().
- procedure, public get_count => percept_predator_get_count_seen

  Get the number (count) of predators seen. See the_neurobio:percept_predator_get_count_seen↩ :().
- procedure, public destroy => percept_predator_destroy_deallocate

  *Deallocate and delete a **predator** perception object. See the_neurobio::percept_predator_destroy_deallocate().*

Public Attributes

- type(spatialobj_percept_comp), dimension(:), allocatable predators_seen

  An array of predators seen in proximity, within the visual range.
- real(srp), dimension(:), allocatable predators_attack_rates

  An array of the attack rates of the predators in the perception object.
- integer predators_seen_count

  The number of conspecifics seen.

9.73.1 Detailed Description

This type defines how the agent perceives a predator.
Definition at line 246 of file m_neuro.f90.

9.73.2 Member Function/Subroutine Documentation

9.73.2.1 init()

`procedure, public the_neurobio::percept_predator::init`
Create **conspecifics** perception object, it is an array of conspecific perception components. See the_neurobio::percept_pre
Definition at line 259 of file m_neuro.f90.

9.73.2.2 number()

`procedure, public the_neurobio::percept_predator::number`
Set the total number of **predators** perceived (seen) in the predator perception object. See the_neurobio::percept_predato
Definition at line 263 of file m_neuro.f90.

9.73.2.3 make()

`procedure, public the_neurobio::percept_predator::make`
Make the **predator** perception object, fill it with the actual arrays. See the_neurobio::percept_predator_make_fill_ar
Definition at line 267 of file m_neuro.f90.

9.73.2.4 set_attack_rate_v()

`procedure, public the_neurobio::percept_predator::set_attack_rate_v`
Set an array of the attack rates for the predator perception object. See the_neurobio::percept_predator_set_attack_
Definition at line 270 of file m_neuro.f90.

9.73.2.5 set_attack_rate_s()

procedure, public the_neurobio::percept_predator::set_attack_rate_s
Set an array of the attack rates for the predator perception object. See the_neurobio::percept_predator_set_attack_
Definition at line 273 of file m_neuro.f90.

9.73.2.6 set_attack_rate()

generic, public the_neurobio::percept_predator::set_attack_rate
A generic interface to set the attack rates for the predator perception object. See the_neurobio::percept_predator_set_a
and the_neurobio::percept_predator_set_attack_rate_scalar().
Definition at line 278 of file m_neuro.f90.

9.73.2.7 get_count()

procedure, public the_neurobio::percept_predator::get_count
Get the number (count) of predators seen. See the_neurobio:percept_predator_get_count_↩
seen:().
Definition at line 281 of file m_neuro.f90.

9.73.2.8 destroy()

procedure, public the_neurobio::percept_predator::destroy
Deallocate and delete a **predator** perception object. See the_neurobio::percept_predator_destroy_deallocate()
Definition at line 284 of file m_neuro.f90.

9.73.3 Member Data Documentation

9.73.3.1 predators_seen

type(spatialobj_percept_comp), dimension(:), allocatable the_neurobio::percept_predator↩
::predators_seen
An array of predators seen in proximity, within the visual range.

Note

Perception of an array of predators uses the arbitrary spatial object components type defined by SPATIALOBJ_PERCEPT_COMP.

Definition at line 250 of file m_neuro.f90.

9.73.3.2 predators_attack_rates

real(srp), dimension(:), allocatable the_neurobio::percept_predator::predators_attack_rates
An array of the attack rates of the predators in the perception object.
Definition at line 252 of file m_neuro.f90.

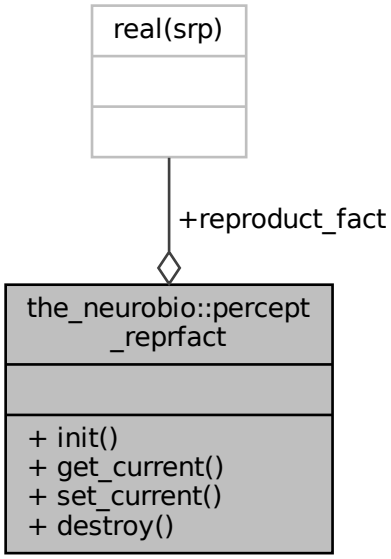
9.73.3.3 predators_seen_count

integer the_neurobio::percept_predator::predators_seen_count
The number of conspecifics seen.
Definition at line 254 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

9.74 the_neurobio::percept_reprfact Type Reference

Perception of the reproductive factor, reproductive factor depends on the sex hormones differently in males and females.
Collaboration diagram for the_neurobio::percept_reprfact:



Public Member Functions

- procedure, public init => percept_reprfac_create_init

  Make en empty reproductive factor perception component. See `the_neurobio::percept_reprfac_create_init()`.
- procedure, public get_current => percept_reprfac_get_current

  Get the current perception of the ***reproductive factor****. See* `the_neurobio::percept_reprfac_get_current()`.
- procedure, public set_current => percept_reprfac_set_current

  Set the current ***reproductive factor*** *level into perception component. See* `the_neurobio::percept_reprfac_set_current`
- procedure, public destroy => percept_reprfac_destroy_deallocate

  Destroy / deallocate ***reproductive factor*** *perception component. See* `the_neurobio::percept_reprfac_destroy_deallo`

Public Attributes

- real(srp) reproduct_fact

9.74.1 Detailed Description

Perception of the reproductive factor, reproductive factor depends on the sex hormones differently in males and females.
Definition at line 374 of file m_neuro.f90.

9.74.2 Member Function/Subroutine Documentation

9.74.2.1 init()

`procedure, public the_neurobio::percept_reprfact::init`
Make en empty reproductive factor perception component. See `the_neurobio::percept_reprfac_create_init()`.
Definition at line 379 of file m_neuro.f90.

9.74.2.2 get_current()

`procedure, public the_neurobio::percept_reprfact::get_current`
Get the current perception of the **reproductive factor**. See `the_neurobio::percept_reprfac_get_current()`.
Definition at line 382 of file m_neuro.f90.

9.74.2.3 set_current()

`procedure, public the_neurobio::percept_reprfact::set_current`
Set the current **reproductive factor** level into perception component. See `the_neurobio::percept_reprfac_set_curre`
Definition at line 385 of file m_neuro.f90.

9.74.2.4 destroy()

`procedure, public the_neurobio::percept_reprfact::destroy`
Destroy / deallocate **reproductive factor** perception component. See `the_neurobio::percept_reprfac_destroy_deal`
Definition at line 388 of file m_neuro.f90.

9.74.3 Member Data Documentation

9.74.3.1 reproduct_fact

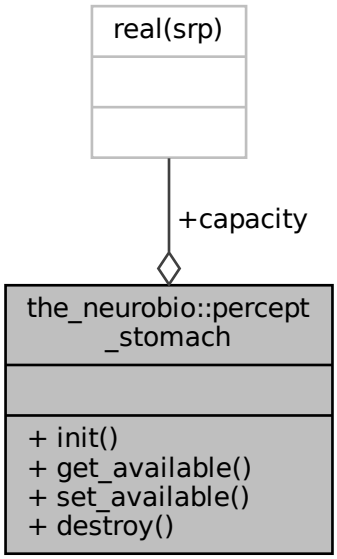
`real(srp) the_neurobio::percept_reprfact::reproduct_fact`
Definition at line 375 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

9.75 the_neurobio::percept_stomach Type Reference

This type defines how the agent perceives its own stomach capacity.

Collaboration diagram for the_neurobio::percept_stomach:



Public Member Functions

- procedure, public init => percept_stomach_create_init

  *Initiate an empty **stomach** capacity perception object. See the_neurobio::percept_stomach_create_init().*
- procedure, public get_available => percept_stomach_get_avail_capacity

  *Get the currently available value of the available **stomach** volume. See the_neurobio::percept_stomach_get_avail_capa*
- procedure, public set_available =>percept_stomach_update_avail_capacity

  *Set and update the currently available value of the available **stomach** volume. See the_neurobio::percept_stomach_update*
- procedure, public destroy => percept_stomach_destroy_deallocate

  *Destroy the **stomach** perception object and deallocate. See the_neurobio::percept_stomach_destroy_deallocate().*

Public Attributes

- real(srp) capacity

  Available stomach capacity as a proportion of the full stomach. So, 0 is full stomach (no space for new food), 1 is empty stomach (full capacity available).

9.75.1 Detailed Description

This type defines how the agent perceives its own stomach capacity.
Definition at line 292 of file m_neuro.f90.

9.75.2 Member Function/Subroutine Documentation

9.75.2.1 init()

`procedure, public the_neurobio::percept_stomach::init`
Initiate an empty **stomach** capacity perception object. See `the_neurobio::percept_stomach_create_init()`.
Definition at line 300 of file m_neuro.f90.

9.75.2.2 get_available()

`procedure, public the_neurobio::percept_stomach::get_available`
Get the currently available value of the available **stomach** volume. See `the_neurobio::percept_stomach_get_avail_ca`
Definition at line 303 of file m_neuro.f90.

9.75.2.3 set_available()

`procedure, public the_neurobio::percept_stomach::set_available`
Set and update the currently available value of the available **stomach** volume. See `the_neurobio::percept_stomach_upda`
Definition at line 307 of file m_neuro.f90.

9.75.2.4 destroy()

`procedure, public the_neurobio::percept_stomach::destroy`
Destroy the **stomach** perception object and deallocate. See `the_neurobio::percept_stomach_destroy_deallocate`
Definition at line 310 of file m_neuro.f90.

9.75.3 Member Data Documentation

9.75.3.1 capacity

`real(srp) the_neurobio::percept_stomach::capacity`
Available stomach capacity as a proportion of the full stomach. So, 0 is full stomach (no space for new food), 1 is empty stomach (full capacity available).
Definition at line 296 of file m_neuro.f90.
The documentation for this type was generated from the following file:

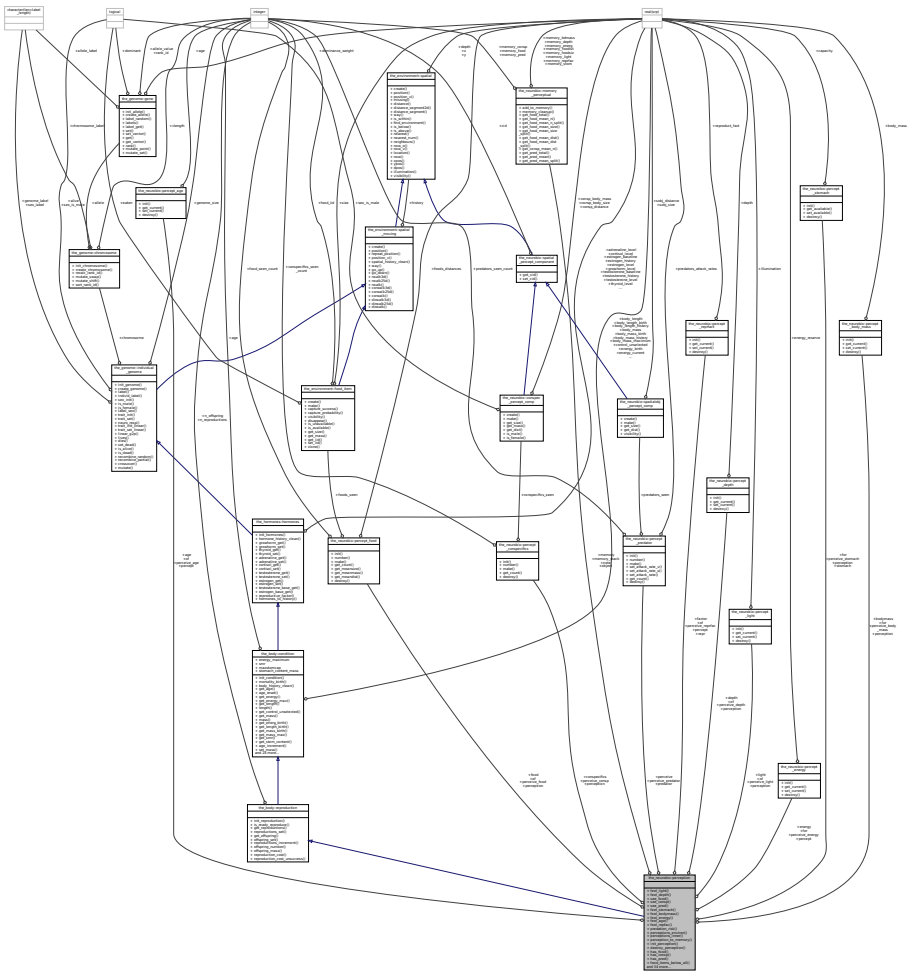
- m_neuro.f90

9.76 the_neurobio::perception Type Reference

The perception architecture of the agent. See "The perception mechanism" for a general overview. At this level, lower order perception objects are combined into the the_neurobio::perception class hierarchy level of the agent. The object bound functions see_ and feel_ obtain (**set**) the specific perception objects from the external or internal environments of the agent and put them into the the_neurobio::perception data structure. Also, memory component is updated with the perception data. Perception objects can then be used as input into the individual decision-making procedures.

Inheritance diagram for the_neurobio::perception:

Collaboration diagram for the_neurobio::perception:



Public Member Functions

- procedure, public feel_light => light_perception_get_object

  *Get **light** perception objects into the individual the_neurobio::perception object layer. See* `the_neurobio::light_perception_g`

- procedure, public feel_depth => depth_perception_get_object

  *Get **depth** perception objects into the **individual** the_neurobio::perception object layer. See* `the_neurobio::depth_perception`

- procedure, public see_food => food_perception_get_visrange_objects

  Get available food items within the visual range of the agent, which the agent can perceive and therefore respond to. Food perception is packaged into the food perception object `thisperceive_food` *for output. See* `the_neurobio::food_perception_get_visrange_objects()`.

- procedure, public see_consp => consp_perception_get_visrange_objects

  Get available conspecific perception objects within the visual range of the agent, which the agent can perceive and therefore respond to. See `the_neurobio::consp_perception_get_visrange_objects()`.

- procedure, public see_pred => predator_perception_get_visrange_objects

  Get available predators perception objects within the visual range of the agent, which the agent can perceive and therefore respond to. See `the_neurobio::predator_perception_get_visrange_objects()`.

- procedure, public feel_stomach => stomach_perception_get_object

  *Get the **stomach capacity** perception objects into the **individual** the_neurobio::perception object layer. See* `the_neurobio::stomach_perception_get_object()`.

- procedure, public feel_bodymass => bodymass_perception_get_object

  *Get the **body mass** perception objects into the **individual** the_neurobio::perception object layer. See* `the_neurobio::bodymass_perception_get_object()`.

- procedure, public feel_energy => energy_perception_get_object

  *Get the **energy reserves** perception objects into the **individual** the_neurobio::perception object layer. See `the_neurobio::energy_perception_get_object()`.*
- procedure, public feel_age => age_perception_get_object

  *Get the **age** perception objects into the **individual** the_neurobio::perception object layer. See `the_neurobio::age_perception_`*
- procedure, public feel_repfac => repfac_perception_get_object

  *Get the **reproductive factor** perception objects into the **individual** the_neurobio::perception object layer. See `the_neurobio::repfac_perception_get_object()`.*
- procedure, public predation_risk => perception_predation_risk_objective

  *Calculate the risk of **predation** as being **perceived** / **assessed** by this agent. See `the_neurobio::perception_predation_ri`*
- procedure, public perceptions_environ => perception_objects_get_all_environmental

  *A single umbrella subroutine to get all **environmental** perceptions: light, depth. See `the_neurobio::perception_objects_ge`*
- procedure, public perceptions_inner => perception_objects_get_all_inner

  *A single umbrella subroutine wrapper to get all **inner** perceptions: stomach, body mass, energy, age. See `the_neurobio::perception_objects_get_all_inner()`.*
- procedure, public perception_to_memory => perception_objects_add_memory_stack

  *Add the various perception objects to the memory stack object. This procedure is called **after** all the perceptual components (light, depth food, conspecifics, predators, etc.) are collected (using `set` object-bound subroutines) into the perception bundle, so all the values are known and ready to be used. See `the_neurobio::perception_objects_add_memory_stack()`.*
- procedure, public init_perception => perception_objects_init_agent

  Initialise all the perception objects for the current agent. Do not fill perception objects with the real data yet. See `the_neurobio::perception_objects_init_agent()`.
- procedure, public destroy_perception => perception_objects_destroy

  Destroy and deallocate all perception objects. See `the_neurobio::perception_objects_destroy()`.
- procedure, public has_food => food_perception_is_seeing_food

  Check if the agent sees any food items within its visual range. See `the_neurobio::food_perception_is_seeing_food()`.
- procedure, public has_consp => consp_perception_is_seeing_conspecifics

  Check if the agent sees any conspecifics within the visual range. See `the_neurobio::consp_perception_is_seeing_cons`
- procedure, public has_pred => predator_perception_is_seeing_predators

  Check if the agent sees any predators within the visual range. See `the_neurobio::predator_perception_is_seeing_pre`
- procedure, public food_items_below_all => perception_food_items_below_calculate

  *Calculate the number of food items in the perception object that are located **below** the actor agent. See `the_neurobio::perception_food_items_below_calculate()`*
- procedure, public food_items_below_horiz => perception_food_items_below_horiz_calculate

  *Calculate the number of food items in the perception object that are located **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z+hz_lower, z+hz_upper]. See `the_neurobio::perception_food_items_below_horiz_calculate()`.*
- generic, public food_items_below => food_items_below_all, food_items_below_horiz

  *A generic interface for the two functions calculating the number of food items in the perception object that are located **below** the actor agent. See perception::food_items_below_all(), perception::food_items_below_horiz().*
- procedure, public food_mass_below_all => perception_food_mass_below_calculate

  *Calculate the average mass of a food item from all the items in the current perception object that are **below** the actor agent. See `the_neurobio::perception_food_mass_below_calculate()`.*
- procedure, public food_mass_below_horiz => perception_food_mass_below_horiz_calculate

  *Calculate the average mass of a food item from all the items in the current perception object that are **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z+hz_lower, z+hz_upper]. See `the_neurobio::perception_food_mass_below_horiz_calculate()`.*
- generic, public food_mass_below => food_mass_below_all, food_mass_below_horiz

  *A generic interface to the two functions that calculating the average mass of food items in the perception object that are located **below** the actor agent. See perception::food_mass_below_all(), perception::food_mass_below_horiz().*
- procedure, public food_items_above_all => perception_food_items_above_calculate

*Calculate the number of food items in the perception object that are located **above** the actor agent. See* `the_neurobio::perception_food_items_above_calculate()`

- procedure, public food_items_above_horiz => perception_food_items_above_horiz_calculate

  *Calculate the number of food items in the perception object that are located **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the* `this` *actor agent: [z-hz_upper, z-hz_upper]. See* `the_neurobio::perception_food_items_above_horiz_calculate()`.
- generic, public food_items_above => food_items_above_all, food_items_above_horiz

  *A generic interface for the two functions calculating the number of food itemsin the perception object that are located **below** the actor agent. See perception::food_items_above_all(), perception::food_items_above_horiz().*
- procedure, public food_mass_above_all => perception_food_mass_above_calculate

  *Calculate the average mass of a food item from all the items in the current perception object that are **above** the actor agent. See* `the_neurobio::perception_food_mass_above_calculate()`.
- procedure, public food_mass_above_horiz => perception_food_mass_above_horiz_calculate

  *Calculate the average mass of a food item from all the items in the current perception object that are **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the* `this` *actor agent: [z-hz_upper, z-hz_upper]. See* `the_neurobio::perception_food_mass_above_horiz_calculate()`.
- generic, public food_mass_above => food_mass_above_all, food_mass_above_horiz

  *A generic interface to the two functions that calculating the average mass of food items in the perception object that are located **above** the actor agent. See perception::food_mass_above_all(), perception::food_mass_above_horiz().*
- procedure, public food_dist_below => perception_food_dist_below_calculate

  *Calculate the average distance to all food items in the current perception object that are **below** the actor agent. See* `the_neurobio::perception_food_dist_below_calculate()`.
- procedure, public food_dist_above => perception_food_dist_above_calculate

  *Calculate the average distance to all food items in the current perception object that are **above** the actor agent. See* `the_neurobio::perception_food_dist_above_calculate()`.
- procedure, public consp_below_all => perception_conspecifics_below_calculate

  *Calculate the number of conspecifics in the perception object that are located **below** the actor agent. See* `the_neurobio::perception_conspecifics_below_calculate()`.
- procedure, public consp_above_all => perception_conspecifics_above_calculate

  *Calculate the number of conspecifics in the perception object that are located **above** the actor agent. See* `the_neurobio::perception_conspecifics_above_calculate()`.
- procedure, public consp_below_horiz => perception_conspecifics_below_horiz_calculate

  *Calculate the number of conspecifics in the perception object that are located **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. See* `the_neurobio::perception_conspecifics_below_horiz_calculate()`.
- procedure, public consp_above_horiz => perception_conspecifics_above_horiz_calculate

  *Calculate the number of conspecifics in the perception object that are located **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. See* `the_neurobio::perception_conspecifics_above_horiz_calculate()`.
- generic, public consp_below => consp_below_all, consp_below_horiz

  *A generic interface to the two functions that calculating the number of conspecifics in the perception object that are located **below** the actor agent. See perception::consp_below_all(), perception::consp_below_horiz().*
- generic, public consp_above => consp_above_all, consp_above_horiz

  *A generic interface to the two functions that calculating the number of conspecifics in the perception object that are located **above** the actor agent. See perception::consp_above_all(), perception::consp_above_horiz().*
- procedure, public consp_dist_below => perception_consp_dist_below_calculate

  *Calculate the average distance to all conspecifics in the current perception object that are **below** the actor agent. See* `the_neurobio::perception_consp_dist_below_calculate()`.
- procedure, public consp_dist_above => perception_consp_dist_above_calculate

  *Calculate the average distance to all conspecifics in the current perception object that are **above** the actor agent. See* `the_neurobio::perception_consp_dist_above_calculate()`.
- procedure, public pred_below_all => perception_predator_below_calculate

  *Calculate the number of predators in the perception object that are located **below** the actor agent. See* `the_neurobio::perception_predator_below_calculate()`.
- procedure, public pred_above_all => perception_predator_above_calculate

*Calculate the number of predators in the perception object that are located **above** the actor agent. See `the_neurobio::perception_predator_above_calculate()`.*

- procedure, public pred_below_horiz => perception_predator_below_horiz_calculate

  *Calculate the number of predators in the perception object that are located **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. See `the_neurobio::perception_predator_below_horiz_calculate`.*

- procedure, public pred_above_horiz => perception_predator_above_horiz_calculate

  *Calculate the number of predators in the perception object that are located **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. See `the_neurobio::perception_predator_above_horiz_calculate`.*

- generic, public pred_below => pred_below_all, pred_below_horiz

  *A generic interface to the two functions that calculating the number of predators in the perception object that are located **below** the actor agent. See perception::pred_below_all(), perception::pred_below_horiz().*

- generic, public pred_above => pred_above_all, pred_above_horiz

  *A generic interface to the two functions that calculating the number of predators in the perception object that are located **above** the actor agent. See perception::pred_above_all(), perception::pred_above_horiz().*

- procedure, public pred_dist_below => perception_predator_dist_below_calculate

  *Calculate the average distance to all predators in the current perception object that are **below** the actor agent. See `the_neurobio::perception_predator_dist_below_calculate()`.*

- procedure, public pred_dist_above => perception_predator_dist_above_calculate

  *Calculate the average distance to all predators in the current perception object that are **above** the actor agent. See `the_neurobio::perception_predator_dist_above_calculate()`.*

- procedure, public risk_pred_s => predator_capture_probability_calculate_spatobj

  Calculate the probability of attack and capture of the `this` agent by the predator `this_predator`. This probability is a function of the distance between the predator and the agent and is calculated by the predator-class-bound procedure the_environment::predator::risk_fish().

- procedure, public risk_pred_p => predator_capture_probability_calculate_pred

  Calculate the probability of attack and capture of the `this` agent by the predator `this_predator`. This probability is a function of the distance between the predator and the agent and is calculated by the predator-class-bound procedure the_environment::predator::risk_fish().

- procedure, public risk_pred_w => predation_capture_probability_risk_wrapper

  Calculate the overall direct predation risk for the agent, i.e. the probability of attack and capture by the nearest predator. See `the_neurobio::predation_capture_probability_risk_wrapper()`.

- generic, public risk_pred => risk_pred_s, risk_pred_p, risk_pred_w

  A single generic interface for the calculation of the probability of attack and capture of the `this` agent by a predator. See `the_neurobio::predator_capture_probability_calculate_spatobj()`, `the_neurobio::predator_capture_probability_calculate_pred()` and `the_neurobio::predation_capt`

- procedure, public food_probability_capture_subjective => food_perception_probability_capture_memory_object

  Calculate the probability of capture of a subjective representation of food item based on the data from the perceptual memory stack. See `the_neurobio::food_perception_probability_capture_memory_object()`.

Public Attributes

- type(percept_light) perceive_light
- type(percept_light) perception
- type(percept_light) of
- type(percept_light) light
- type(percept_depth) perceive_depth
- type(percept_depth) perception
- type(percept_depth) of
- type(percept_depth) depth
- type(percept_food) perceive_food
- type(percept_food) perception
- type(percept_food) of
- type(percept_food) food
- type(percept_conspecifics) perceive_consp
- type(percept_conspecifics) conspecifics

- type(percept_conspecifics) perception
- type(percept_predator) perceive_predator
- type(percept_predator) perceive
- type(percept_predator) predator
- type(percept_stomach) perceive_stomach
- type(percept_stomach) perception
- type(percept_stomach) for
- type(percept_stomach) stomach
- type(percept_body_mass) perceive_body_mass
- type(percept_body_mass) perception
- type(percept_body_mass) for
- type(percept_body_mass) bodymass
- type(percept_energy) perceive_energy
- type(percept_energy) percept
- type(percept_energy) for
- type(percept_energy) energy
- type(percept_age) perceive_age
- type(percept_age) percept
- type(percept_age) of
- type(percept_age) age
- type(percept_reprfact) perceive_reprfac
- type(percept_reprfact) percept
- type(percept_reprfact) of
- type(percept_reprfact) repr
- type(percept_reprfact) factor
- type(memory_perceptual) memory_stack
- type(memory_perceptual) note
- type(memory_perceptual) memory
- type(memory_perceptual) object

9.76.1 Detailed Description

The perception architecture of the agent. See "The perception mechanism" for a general overview. At this level, lower order perception objects are combined into the the_neurobio::perception class hierarchy level of the agent. The object bound functions see_ and feel_ obtain (**set**) the specific perception objects from the external or internal environments of the agent and put them into the the_neurobio::perception data structure. Also, memory component is updated with the perception data. Perception objects can then be used as input into the individual decision-making procedures.

Note

Templates for outer environmental perceptions:

```
call proto_parents%individual(ind)%see_food(                    &
            food_resource_available = habitat_safe%food,        &
            time_step_model = 1)
call proto_parents%individual(ind)%see_consp(                   &
            consp_agents = proto_parents%individual,            &
            time_step_model = 1 )
call proto_parents%individual(ind)%see_pred(                    &
            spatl_agents = predators,                           &
            time_step_model = 1 )
call proto_parents%individual(ind)%feel_light(timestep)`
call proto_parents%individual(ind)%feel_depth()
```

Definition at line 561 of file m_neuro.f90.

9.76.2 Member Function/Subroutine Documentation

9.76.2.1 feel_light()

`procedure, public the_neurobio::perception::feel_light`
Get **light** perception objects into the individual the_neurobio::perception object layer. See `the_neurobio::light_perceptio`
Definition at line 577 of file m_neuro.f90.

9.76.2.2 feel_depth()

`procedure, public the_neurobio::perception::feel_depth`
Get **depth** perception objects into the **individual** the_neurobio::perception object layer. See `the_neurobio::depth_percepti`
Definition at line 581 of file m_neuro.f90.

9.76.2.3 see_food()

`procedure, public the_neurobio::perception::see_food`
Get available food items within the visual range of the agent, which the agent can perceive and therefore respond to. Food perception is packaged into the food perception object `thisperceive_food` for output. See `the_neurobio::food_perception_get_visrange_objects()`.
Definition at line 586 of file m_neuro.f90.

9.76.2.4 see_consp()

`procedure, public the_neurobio::perception::see_consp`
Get available conspecific perception objects within the visual range of the agent, which the agent can perceive and therefore respond to. See `the_neurobio::consp_perception_get_visrange_objects()`.
Definition at line 590 of file m_neuro.f90.

9.76.2.5 see_pred()

`procedure, public the_neurobio::perception::see_pred`
Get available predators perception objects within the visual range of the agent, which the agent can perceive and therefore respond to. See `the_neurobio::predator_perception_get_visrange_objects()`.
Definition at line 594 of file m_neuro.f90.

9.76.2.6 feel_stomach()

`procedure, public the_neurobio::perception::feel_stomach`
Get the **stomach capacity** perception objects into the **individual** the_neurobio::perception object layer. See `the_neurobio::stomach_perception_get_object()`.
Definition at line 598 of file m_neuro.f90.

9.76.2.7 feel_bodymass()

`procedure, public the_neurobio::perception::feel_bodymass`
Get the **body mass** perception objects into the **individual** the_neurobio::perception object layer. See `the_neurobio::bodymass_perception_get_object()`.
Definition at line 602 of file m_neuro.f90.

9.76.2.8 feel_energy()

`procedure, public the_neurobio::perception::feel_energy`
Get the **energy reserves** perception objects into the **individual** the_neurobio::perception object layer. See `the_neurobio::energy_perception_get_object()`.

Definition at line 606 of file m_neuro.f90.

9.76.2.9 feel_age()

`procedure, public the_neurobio::perception::feel_age`
Get the **age** perception objects into the **individual** the_neurobio::perception object layer. See `the_neurobio::age_perceptio`
Definition at line 610 of file m_neuro.f90.

9.76.2.10 feel_repfac()

`procedure, public the_neurobio::perception::feel_repfac`
Get the **reproductive factor** perception objects into the **individual** the_neurobio::perception object layer. See `the_neurobio::repfac_perception_get_object()`.
Definition at line 614 of file m_neuro.f90.

9.76.2.11 predation_risk()

`procedure, public the_neurobio::perception::predation_risk`
Calculate the risk of **predation** as being **perceived** / **assessed** by this agent. See `the_neurobio::perception_predation_`
Definition at line 619 of file m_neuro.f90.

9.76.2.12 perceptions_environ()

`procedure, public the_neurobio::perception::perceptions_environ`
A single umbrella subroutine to get all **environmental** perceptions: light, depth. See `the_neurobio::perception_objects_`
Definition at line 623 of file m_neuro.f90.

9.76.2.13 perceptions_inner()

`procedure, public the_neurobio::perception::perceptions_inner`
A single umbrella subroutine wrapper to get all **inner** perceptions: stomach, body mass, energy, age. See `the_neurobio::perception_objects_get_all_inner()`.
Definition at line 627 of file m_neuro.f90.

9.76.2.14 perception_to_memory()

`procedure, public the_neurobio::perception::perception_to_memory`
Add the various perception objects to the memory stack object. This procedure is called **after** all the perceptual components (light, depth food, conspecifics, predators, etc.) are collected (using `set` object-bound subroutines) into the perception bundle, so all the values are known and ready to be used. See `the_neurobio::perception_objects_add_memory_stack()`.
Definition at line 634 of file m_neuro.f90.

9.76.2.15 init_perception()

`procedure, public the_neurobio::perception::init_perception`
Initialise all the perception objects for the current agent. Do not fill perception objects with the real data yet. See `the_neurobio::perception_objects_init_agent()`.
Definition at line 639 of file m_neuro.f90.

9.76.2.16 destroy_perception()

`procedure, public the_neurobio::perception::destroy_perception`
Destroy and deallocate all perception objects. See `the_neurobio::perception_objects_destroy()`.
Definition at line 642 of file m_neuro.f90.

9.76.2.17 has_food()

`procedure, public the_neurobio::perception::has_food`
Check if the agent sees any food items within its visual range. See `the_neurobio::food_perception_is_seeing_food`
Definition at line 647 of file m_neuro.f90.

9.76.2.18 has_consp()

`procedure, public the_neurobio::perception::has_consp`
Check if the agent sees any conspecifics within the visual range. See `the_neurobio::consp_perception_is_seeing_co`
Definition at line 650 of file m_neuro.f90.

9.76.2.19 has_pred()

`procedure, public the_neurobio::perception::has_pred`
Check if the agent sees any predators within the visual range. See `the_neurobio::predator_perception_is_seeing_p`
Definition at line 653 of file m_neuro.f90.

9.76.2.20 food_items_below_all()

`procedure, public the_neurobio::perception::food_items_below_all`
Calculate the number of food items in the perception object that are located **below** the actor agent. See `the_neurobio::perception_food_items_below_calculate()`
Definition at line 659 of file m_neuro.f90.

9.76.2.21 food_items_below_horiz()

`procedure, public the_neurobio::perception::food_items_below_horiz`
Calculate the number of food items in the perception object that are located **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z+hz_lower, z+hz_upper]. See `the_neurobio::perception_food_items_below_horiz_calcula`
Definition at line 667 of file m_neuro.f90.

9.76.2.22 food_items_below()

`generic, public the_neurobio::perception::food_items_below`
A generic interface for the two functions calculating the number of food items in the perception object that are located **below** the actor agent. See perception::food_items_below_all(), perception::food_items_below_horiz().
Definition at line 673 of file m_neuro.f90.

9.76.2.23 food_mass_below_all()

`procedure, public the_neurobio::perception::food_mass_below_all`
Calculate the average mass of a food item from all the items in the current perception object that are **below** the actor agent. See `the_neurobio::perception_food_mass_below_calculate()`.
Definition at line 678 of file m_neuro.f90.

9.76.2.24 food_mass_below_horiz()

procedure, public the_neurobio::perception::food_mass_below_horiz
Calculate the average mass of a food item from all the items in the current perception object that are **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the this actor agent: [z+hz_lower, z+hz_upper]. See the_neurobio::perception_food_mass_below_horiz_calculate().
Definition at line 686 of file m_neuro.f90.

9.76.2.25 food_mass_below()

generic, public the_neurobio::perception::food_mass_below
A generic interface to the two functions that calculating the average mass of food items in the perception object that are located **below** the actor agent. See perception::food_mass_below_all(), perception::food_mass_below_horiz().
Definition at line 692 of file m_neuro.f90.

9.76.2.26 food_items_above_all()

procedure, public the_neurobio::perception::food_items_above_all
Calculate the number of food items in the perception object that are located **above** the actor agent. See the_neurobio::perception_food_items_above_calculate()
Definition at line 697 of file m_neuro.f90.

9.76.2.27 food_items_above_horiz()

procedure, public the_neurobio::perception::food_items_above_horiz
Calculate the number of food items in the perception object that are located **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the this actor agent: [z-hz_upper, z-hz_upper]. See the_neurobio::perception_food_items_above_horiz_calcula
Definition at line 705 of file m_neuro.f90.

9.76.2.28 food_items_above()

generic, public the_neurobio::perception::food_items_above
A generic interface for the two functions calculating the number of food itemsin the perception object that are located **below** the actor agent. See perception::food_items_above_all(), perception::food_items_above_horiz().
Definition at line 711 of file m_neuro.f90.

9.76.2.29 food_mass_above_all()

procedure, public the_neurobio::perception::food_mass_above_all
Calculate the average mass of a food item from all the items in the current perception object that are **above** the actor agent. See the_neurobio::perception_food_mass_above_calculate().
Definition at line 716 of file m_neuro.f90.

9.76.2.30 food_mass_above_horiz()

procedure, public the_neurobio::perception::food_mass_above_horiz
Calculate the average mass of a food item from all the items in the current perception object that are **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the this actor agent: [z-hz_upper, z-hz_upper]. See the_neurobio::perception_food_mass_above_horiz_calculate().
Definition at line 724 of file m_neuro.f90.

9.76.2.31 food_mass_above()

`generic, public the_neurobio::perception::food_mass_above`
A generic interface to the two functions that calculating the average mass of food items in the perception object that are located **above** the actor agent. See perception::food_mass_above_all(), perception::food_mass_above_horiz().
Definition at line 730 of file m_neuro.f90.

9.76.2.32 food_dist_below()

`procedure, public the_neurobio::perception::food_dist_below`
Calculate the average distance to all food items in the current perception object that are **below** the actor agent. See `the_neurobio::perception_food_dist_below_calculate()`.
Definition at line 735 of file m_neuro.f90.

9.76.2.33 food_dist_above()

`procedure, public the_neurobio::perception::food_dist_above`
Calculate the average distance to all food items in the current perception object that are **above** the actor agent. See `the_neurobio::perception_food_dist_above_calculate()`.
Definition at line 740 of file m_neuro.f90.

9.76.2.34 consp_below_all()

`procedure, public the_neurobio::perception::consp_below_all`
Calculate the number of conspecifics in the perception object that are located **below** the actor agent. See `the_neurobio::perception_conspecifics_below_calculate()`.
Definition at line 747 of file m_neuro.f90.

9.76.2.35 consp_above_all()

`procedure, public the_neurobio::perception::consp_above_all`
Calculate the number of conspecifics in the perception object that are located **above** the actor agent. See `the_neurobio::perception_conspecifics_above_calculate()`.
Definition at line 752 of file m_neuro.f90.

9.76.2.36 consp_below_horiz()

`procedure, public the_neurobio::perception::consp_below_horiz`
Calculate the number of conspecifics in the perception object that are located **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. See `the_neurobio::perception_conspecifics_below_horiz_calculate(`
Definition at line 758 of file m_neuro.f90.

9.76.2.37 consp_above_horiz()

`procedure, public the_neurobio::perception::consp_above_horiz`
Calculate the number of conspecifics in the perception object that are located **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. See `the_neurobio::perception_conspecifics_above_horiz_calculate(`
Definition at line 764 of file m_neuro.f90.

9.76.2.38 consp_below()

`generic, public the_neurobio::perception::consp_below`

A generic interface to the two functions that calculating the number of conspecifics in the perception object that are located **below** the actor agent. See perception::consp_below_all(), perception::consp_below_horiz().
Definition at line 770 of file m_neuro.f90.

9.76.2.39 consp_above()

`generic, public the_neurobio::perception::consp_above`
A generic interface to the two functions that calculating the number of conspecifics in the perception object that are located **above** the actor agent. See perception::consp_above_all(), perception::consp_above_horiz().
Definition at line 775 of file m_neuro.f90.

9.76.2.40 consp_dist_below()

`procedure, public the_neurobio::perception::consp_dist_below`
Calculate the average distance to all conspecifics in the current perception object that are **below** the actor agent. See `the_neurobio::perception_consp_dist_below_calculate()`.
Definition at line 779 of file m_neuro.f90.

9.76.2.41 consp_dist_above()

`procedure, public the_neurobio::perception::consp_dist_above`
Calculate the average distance to all conspecifics in the current perception object that are **above** the actor agent. See `the_neurobio::perception_consp_dist_above_calculate()`.
Definition at line 784 of file m_neuro.f90.

9.76.2.42 pred_below_all()

`procedure, public the_neurobio::perception::pred_below_all`
Calculate the number of predators in the perception object that are located **below** the actor agent. See `the_neurobio::perception_predator_below_calculate()`.
Definition at line 791 of file m_neuro.f90.

9.76.2.43 pred_above_all()

`procedure, public the_neurobio::perception::pred_above_all`
Calculate the number of predators in the perception object that are located **above** the actor agent. See `the_neurobio::perception_predator_above_calculate()`.
Definition at line 795 of file m_neuro.f90.

9.76.2.44 pred_below_horiz()

`procedure, public the_neurobio::perception::pred_below_horiz`
Calculate the number of predators in the perception object that are located **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. See `the_neurobio::perception_predator_below_horiz_calculate`.
Definition at line 800 of file m_neuro.f90.

9.76.2.45 pred_above_horiz()

`procedure, public the_neurobio::perception::pred_above_horiz`
Calculate the number of predators in the perception object that are located **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. See `the_neurobio::perception_predator_above_horiz_calculate`.
Definition at line 806 of file m_neuro.f90.

9.76.2.46 pred_below()

`generic, public the_neurobio::perception::pred_below`
A generic interface to the two functions that calculating the number of predators in the perception object that are located **below** the actor agent. See perception::pred_below_all(), perception::pred_below_horiz().
Definition at line 812 of file m_neuro.f90.

9.76.2.47 pred_above()

`generic, public the_neurobio::perception::pred_above`
A generic interface to the two functions that calculating the number of predators in the perception object that are located **above** the actor agent. See perception::pred_above_all(), perception::pred_above_horiz().
Definition at line 817 of file m_neuro.f90.

9.76.2.48 pred_dist_below()

`procedure, public the_neurobio::perception::pred_dist_below`
Calculate the average distance to all predators in the current perception object that are **below** the actor agent. See `the_neurobio::perception_predator_dist_below_calculate()`.
Definition at line 821 of file m_neuro.f90.

9.76.2.49 pred_dist_above()

`procedure, public the_neurobio::perception::pred_dist_above`
Calculate the average distance to all predators in the current perception object that are **above** the actor agent. See `the_neurobio::perception_predator_dist_above_calculate()`.
Definition at line 826 of file m_neuro.f90.

9.76.2.50 risk_pred_s()

`procedure, public the_neurobio::perception::risk_pred_s`
Calculate the probability of attack and capture of the `this` agent by the predator `this_predator`. This probability is a function of the distance between the predator and the agent and is calculated by the predator-class-bound procedure the_environment::predator::risk_fish().

Note

Note that this version of the procedure accepts `this_predator` parameter as class the_neurobio::spatialobj_percept_comp that is used for keeping the predator representations in the **perception object**. This representation keeps two separate array for the_neurobio::spatialobj_percept_comp spatial objects and the attack rate. See `the_neurobio::predator_capture_probability_calculate_spatobj()`.

Definition at line 839 of file m_neuro.f90.

9.76.2.51 risk_pred_p()

`procedure, public the_neurobio::perception::risk_pred_p`
Calculate the probability of attack and capture of the `this` agent by the predator `this_predator`. This probability is a function of the distance between the predator and the agent and is calculated by the predator-class-bound procedure the_environment::predator::risk_fish().

Note

Note that this version of the procedure accepts `this_predator` parameter as class the_neurobio↩ ::predator, i.e. for the **objective predator object**. See `the_neurobio::predator_capture_probability_calcul`

Definition at line 849 of file m_neuro.f90.

9.76.2.52 risk_pred_w()

procedure, public the_neurobio::perception::risk_pred_w
Calculate the overall direct predation risk for the agent, i.e. the probability of attack and capture by the nearest predator. See the_neurobio::predation_capture_probability_risk_wrapper().
Definition at line 854 of file m_neuro.f90.

9.76.2.53 risk_pred()

generic, public the_neurobio::perception::risk_pred
A single generic interface for the calculation of the probability of attack and capture of the this agent by a predator. See the_neurobio::predator_capture_probability_calculate_spatobj(), the_neurobio::predator_capture_probability_calculate_pred() and the_neurobio::predation_ca
Definition at line 861 of file m_neuro.f90.

9.76.2.54 food_probability_capture_subjective()

procedure, public the_neurobio::perception::food_probability_capture_subjective
Calculate the probability of capture of a subjective representation of food item based on the data from the perceptual memory stack. See the_neurobio::food_perception_probability_capture_memory_object().
Definition at line 866 of file m_neuro.f90.

9.76.3 Member Data Documentation

9.76.3.1 perceive_light

type(percept_light) the_neurobio::perception::perceive_light
Definition at line 562 of file m_neuro.f90.

9.76.3.2 perception [1/6]

type(percept_light) the_neurobio::perception::perception
Definition at line 562 of file m_neuro.f90.

9.76.3.3 of [1/5]

type(percept_light) the_neurobio::perception::of
Definition at line 562 of file m_neuro.f90.

9.76.3.4 light

type(percept_light) the_neurobio::perception::light
Definition at line 562 of file m_neuro.f90.

9.76.3.5 perceive_depth

type(percept_depth) the_neurobio::perception::perceive_depth
Definition at line 563 of file m_neuro.f90.

9.76.3.6 perception [2/6]

type(percept_depth) the_neurobio::perception::perception
Definition at line 563 of file m_neuro.f90.

9.76.3.7 of [2/5]

type(percept_depth) the_neurobio::perception::of
Definition at line 563 of file m_neuro.f90.

9.76.3.8 depth

type(percept_depth) the_neurobio::perception::depth
Definition at line 563 of file m_neuro.f90.

9.76.3.9 perceive_food

type(percept_food) the_neurobio::perception::perceive_food
Definition at line 564 of file m_neuro.f90.

9.76.3.10 perception [3/6]

type(percept_food) the_neurobio::perception::perception
Definition at line 564 of file m_neuro.f90.

9.76.3.11 of [3/5]

type(percept_food) the_neurobio::perception::of
Definition at line 564 of file m_neuro.f90.

9.76.3.12 food

type(percept_food) the_neurobio::perception::food
Definition at line 564 of file m_neuro.f90.

9.76.3.13 perceive_consp

type(percept_conspecifics) the_neurobio::perception::perceive_consp
Definition at line 565 of file m_neuro.f90.

9.76.3.14 conspecifics

type(percept_conspecifics) the_neurobio::perception::conspecifics
Definition at line 565 of file m_neuro.f90.

9.76.3.15 perception [4/6]

type(percept_conspecifics) the_neurobio::perception::perception
Definition at line 565 of file m_neuro.f90.

9.76.3.16 perceive_predator

type(percept_predator) the_neurobio::perception::perceive_predator
Definition at line 566 of file m_neuro.f90.

9.76.3.17 perceive

type(percept_predator) the_neurobio::perception::perceive
Definition at line 566 of file m_neuro.f90.

9.76.3.18 predator

type(percept_predator) the_neurobio::perception::predator
Definition at line 566 of file m_neuro.f90.

9.76.3.19 perceive_stomach

type(percept_stomach) the_neurobio::perception::perceive_stomach
Definition at line 567 of file m_neuro.f90.

9.76.3.20 perception [5/6]

type(percept_stomach) the_neurobio::perception::perception
Definition at line 567 of file m_neuro.f90.

9.76.3.21 for [1/3]

type(percept_stomach) the_neurobio::perception::for
Definition at line 567 of file m_neuro.f90.

9.76.3.22 stomach

type(percept_stomach) the_neurobio::perception::stomach
Definition at line 567 of file m_neuro.f90.

9.76.3.23 perceive_body_mass

type(percept_body_mass) the_neurobio::perception::perceive_body_mass
Definition at line 568 of file m_neuro.f90.

9.76.3.24 perception [6/6]

type(percept_body_mass) the_neurobio::perception::perception
Definition at line 568 of file m_neuro.f90.

9.76.3.25 for [2/3]

type(percept_body_mass) the_neurobio::perception::for
Definition at line 568 of file m_neuro.f90.

9.76.3.26 bodymass

type(percept_body_mass) the_neurobio::perception::bodymass
Definition at line 568 of file m_neuro.f90.

9.76.3.27 perceive_energy

type(percept_energy) the_neurobio::perception::perceive_energy
Definition at line 569 of file m_neuro.f90.

9.76.3.28 percept [1/3]

type(percept_energy) the_neurobio::perception::percept
Definition at line 569 of file m_neuro.f90.

9.76.3.29 for [3/3]

type(percept_energy) the_neurobio::perception::for
Definition at line 569 of file m_neuro.f90.

9.76.3.30 energy

type(percept_energy) the_neurobio::perception::energy
Definition at line 569 of file m_neuro.f90.

9.76.3.31 perceive_age

type(percept_age) the_neurobio::perception::perceive_age
Definition at line 570 of file m_neuro.f90.

9.76.3.32 percept [2/3]

type(percept_age) the_neurobio::perception::percept
Definition at line 570 of file m_neuro.f90.

9.76.3.33 of [4/5]

type(percept_age) the_neurobio::perception::of
Definition at line 570 of file m_neuro.f90.

9.76.3.34 age

type(percept_age) the_neurobio::perception::age
Definition at line 570 of file m_neuro.f90.

9.76.3.35 perceive_reprfac

type(percept_reprfact) the_neurobio::perception::perceive_reprfac
Definition at line 571 of file m_neuro.f90.

9.76.3.36 percept [3/3]

type(percept_reprfact) the_neurobio::perception::percept
Definition at line 571 of file m_neuro.f90.

9.76.3.37 of [5/5]

type(percept_reprfact) the_neurobio::perception::of
Definition at line 571 of file m_neuro.f90.

9.76.3.38 repr

type(percept_reprfact) the_neurobio::perception::repr
Definition at line 571 of file m_neuro.f90.

9.76.3.39 factor

type(percept_reprfact) the_neurobio::perception::factor
Definition at line 571 of file m_neuro.f90.

9.76.3.40 memory_stack

type(memory_perceptual) the_neurobio::perception::memory_stack
Definition at line 572 of file m_neuro.f90.

9.76.3.41 note

type(memory_perceptual) the_neurobio::perception::note
Definition at line 572 of file m_neuro.f90.

9.76.3.42 memory

type(memory_perceptual) the_neurobio::perception::memory
Definition at line 572 of file m_neuro.f90.

9.76.3.43 object

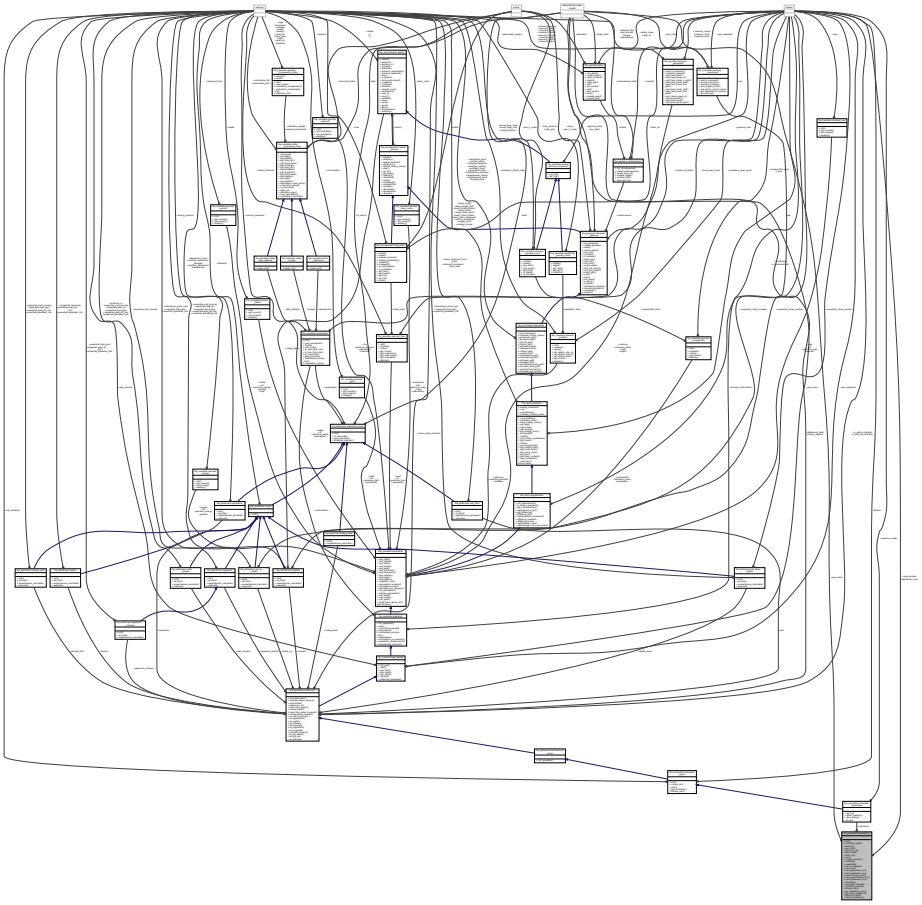
type(memory_perceptual) the_neurobio::perception::object
Definition at line 572 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

9.77 the_population::population Type Reference

Definition of the population object.

Collaboration diagram for the_population::population:



Public Member Functions

- procedure, public init => init_population_random

  Initialise the population object. See `the_population::init_population_random()`.
- procedure, public mortality_birth => population_birth_mortality_init

  Impose selective mortality at birth of the agents. See `the_population::population_birth_mortality_init()`.
- procedure, public destroy => population_destroy_deallocate_objects

  Destroys this population and deallocates the array of individual member objects. See `the_population::population_destroy_`
- procedure, public get_size => population_get_popsize

  Get the size of this population. See `the_population::population_get_popsize()`.
- procedure, public get_num_id => population_get_pop_number

  Get the population number ID. See `the_population::population_get_pop_number()`.
- procedure, public get_name => population_get_pop_name

  Get the population character label ID. See `the_population::population_get_pop_name()`.
- procedure, public reset_id => reset_population_id_random

  Reset individual IDs of the population members. See `the_population::reset_population_id_random()`.
- procedure, public sex => sex_initialise_from_genome

  Determine the sex for each member of the population. See `the_population::sex_initialise_from_genome()`.
- procedure, public scatter_uniform => position_individuals_uniform

  Position each member of the population randomly within a bounding environment with the ***uniform*** *distribution. See* `the_population::position_individuals_uniform()`.
- procedure, public rwalk3d => population_rwalk3d_all_agents_step

  Perform a single step of random walk by all agents, in 3D. See `the_population::population_rwalk3d_all_agents_step`

- procedure, public rwalk25d => population_rwalk25d_all_agents_step

  Perform a single step of random walk by all agents, in 2.5D. See `the_population::population_rwalk25d_all_agents_st`
- procedure, public sort_by_fitness => sort_population_by_fitness

  This subroutine sorts the population `individual` *object by their %fitness components. See* `the_population::sort_populati`
- procedure, public save_csv => population_save_data_all_agents_csv

  Save data for all agents within the population into a csv file. See `the_population::population_save_data_all_agents_c`
- procedure, public save_genomes_csv => population_save_data_all_genomes

  Save the genome data of all agents in this population to a CSV file. See `the_population::population_save_data_all_ger`
- procedure, public load_genomes_csv => population_load_data_all_genomes

  Load agent genome data in this population from a CSV file. See `the_population::population_load_data_all_genomes`
- procedure, public save_memory_csv => population_save_data_memory

  Save the perceptual and emotional memory stack data of all agents in this population to a CSV file. See `the_population::population_save_data_memory()`.
- procedure, public save_movements_csv => population_save_data_movements

  Save the latest movement history of all agents. See `the_population::population_save_data_movements()`.
- procedure, public save_behaviour_csv => population_save_data_behaviours

  Save the behaviours history the_neurobio::behaviour::history_behave for all agents. See `the_population::population_save_`
- procedure, public attacked => population_subject_predator_attack

  Subject the population to an attack by a specific predator. See `the_population::population_subject_predator_attack`
- procedure, public mortality_habitat => population_subject_other_risks

  Subject the population to mortality caused by habitat-specific mortality risk. Each agent is affected by the risk associated with the habitat it is currently in. See `the_population::population_subject_other_risks()`.
- procedure, public mortality_individ => population_subject_individual_risk_mortality

  Subject all members of this population to their individual mortality risks. See `the_population::population_subject_indivi`
- procedure, public fitness_calc => population_preevol_fitness_calc

  Calculate fitness for the pre-evolution phase of the genetic algorithm. ***Pre-evolution*** *is based on selection for a simple criterion without explicit reproduction etc. The criterion for selection at this phase is set by the integer the_individual::individual_agent::fitness component. This procedure provides a whole-population wrapper for the the_individual::fitness_calc() function. See* `the_population::population_preevol_fitness_calc()`.
- procedure, public ga_reproduce_max => population_ga_reproduce_max

  Determine the number of parents that have fitness higher than the minimum acceptable value. See `the_population::population_ga_reproduce_max()`.
- procedure, public ga_mutat_adaptive => population_ga_mutation_rate_adaptive

  This function implements adaptive mutation rate that increases as the population size reduces. See `the_population::population_ga_mutation_rate_adaptive()`.
- procedure, public lifecycle_step => population_lifecycle_step_preevol

  Perform a single step of the life cycle of the population. See `the_population::population_lifecycle_step_preevol()`.
- procedure, public lifecycle_eatonly => population_lifecycle_step_eatonly_preevol

  Perform a single step of the life cycle of the population. This version includes only optimal food selection and eating without the full fledged behaviour selection cascade of procedures ::do_behave(). See `the_population::population_lifecycle_step_eatonly_preevol()`.

Public Attributes

- integer population_size

  The size of the population.
- type(member_population), dimension(:), allocatable individual

  `POPULATION` *is represented by an array of objects of the type* `MEMBER_POPULATION`
- integer pop_number

  The numeric ID of the population (if we have several populations))
- character(len=label_length) pop_name

  The descriptive name of the population.

9.77.1 Detailed Description

Definition of the population object.
This is basically an array of individuals (of the type `MEMBER_POPULATION`) plus population or generation descriptors.
Definition at line 58 of file m_popul.f90.

9.77.2 Member Function/Subroutine Documentation

9.77.2.1 init()

`procedure, public the_population::population::init`
Initialise the population object. See `the_population::init_population_random()`.
Definition at line 77 of file m_popul.f90.

9.77.2.2 mortality_birth()

`procedure, public the_population::population::mortality_birth`
Impose selective mortality at birth of the agents. See `the_population::population_birth_mortality_init()`.
Definition at line 80 of file m_popul.f90.

9.77.2.3 destroy()

`procedure, public the_population::population::destroy`
Destroys this population and deallocates the array of individual member objects. See `the_population::population_destr`
Definition at line 84 of file m_popul.f90.

9.77.2.4 get_size()

`procedure, public the_population::population::get_size`
Get the size of this population. See `the_population::population_get_popsize()`.
Definition at line 87 of file m_popul.f90.

9.77.2.5 get_num_id()

`procedure, public the_population::population::get_num_id`
Get the population number ID. See `the_population::population_get_pop_number()`.
Definition at line 90 of file m_popul.f90.

9.77.2.6 get_name()

`procedure, public the_population::population::get_name`
Get the population character label ID. See `the_population::population_get_pop_name()`.
Definition at line 93 of file m_popul.f90.

9.77.2.7 reset_id()

`procedure, public the_population::population::reset_id`
Reset individual IDs of the population members. See `the_population::reset_population_id_random()`.
Definition at line 96 of file m_popul.f90.

9.77.2.8 sex()

`procedure, public the_population::population::sex`
Determine the sex for each member of the population. See `the_population::sex_initialise_from_genome()`.
Definition at line 99 of file m_popul.f90.

9.77.2.9 scatter_uniform()

`procedure, public the_population::population::scatter_uniform`
Position each member of the population randomly within a bounding environment with the **uniform** distribution. See `the_population::position_individuals_uniform()`.
Definition at line 103 of file m_popul.f90.

9.77.2.10 rwalk3d()

`procedure, public the_population::population::rwalk3d`
Perform a single step of random walk by all agents, in 3D. See `the_population::population_rwalk3d_all_agents_st`
Definition at line 106 of file m_popul.f90.

9.77.2.11 rwalk25d()

`procedure, public the_population::population::rwalk25d`
Perform a single step of random walk by all agents, in 2.5D. See `the_population::population_rwalk25d_all_agents`
Definition at line 109 of file m_popul.f90.

9.77.2.12 sort_by_fitness()

`procedure, public the_population::population::sort_by_fitness`
This subroutine sorts the population `individual` object by their %fitness components. See `the_population::sort_popul`
Definition at line 113 of file m_popul.f90.

9.77.2.13 save_csv()

`procedure, public the_population::population::save_csv`
Save data for all agents within the population into a csv file. See `the_population::population_save_data_all_agent`
Definition at line 116 of file m_popul.f90.

9.77.2.14 save_genomes_csv()

`procedure, public the_population::population::save_genomes_csv`
Save the genome data of all agents in this population to a CSV file. See `the_population::population_save_data_all_`
Definition at line 119 of file m_popul.f90.

9.77.2.15 load_genomes_csv()

`procedure, public the_population::population::load_genomes_csv`
Load agent genome data in this population from a CSV file. See `the_population::population_load_data_all_genom`
Definition at line 122 of file m_popul.f90.

9.77.2.16 save_memory_csv()

procedure, public the_population::population::save_memory_csv
Save the perceptual and emotional memory stack data of all agents in this population to a CSV file. See the_population::population_save_data_memory().
Definition at line 126 of file m_popul.f90.

9.77.2.17 save_movements_csv()

procedure, public the_population::population::save_movements_csv
Save the latest movement history of all agents. See the_population::population_save_data_movements().
Definition at line 129 of file m_popul.f90.

9.77.2.18 save_behaviour_csv()

procedure, public the_population::population::save_behaviour_csv
Save the behaviours history the_neurobio::behaviour::history_behave for all agents. See the_population::population_sav
Definition at line 133 of file m_popul.f90.

9.77.2.19 attacked()

procedure, public the_population::population::attacked
Subject the population to an attack by a specific predator. See the_population::population_subject_predator_att
Definition at line 136 of file m_popul.f90.

9.77.2.20 mortality_habitat()

procedure, public the_population::population::mortality_habitat
Subject the population to mortality caused by habitat-specific mortality risk. Each agent is affected by the risk associated with the habitat it is currently in. See the_population::population_subject_other_risks().
Definition at line 141 of file m_popul.f90.

9.77.2.21 mortality_individ()

procedure, public the_population::population::mortality_individ
Subject all members of this population to their individual mortality risks. See the_population::population_subject_ind
Definition at line 145 of file m_popul.f90.

9.77.2.22 fitness_calc()

procedure, public the_population::population::fitness_calc
Calculate fitness for the pre-evolution phase of the genetic algorithm. **Pre-evolution** is based on selection for a simple criterion without explicit reproduction etc. The criterion for selection at this phase is set by the integer the_individual::individual_agent::fitness component. This procedure provides a whole-population wrapper for the the_individual::fitness_calc() function. See the_population::population_preevol_fitness_calc().
Definition at line 154 of file m_popul.f90.

9.77.2.23 ga_reproduce_max()

procedure, public the_population::population::ga_reproduce_max
Determine the number of parents that have fitness higher than the minimum acceptable value. See the_population::population_ga_reproduce_max().
Definition at line 158 of file m_popul.f90.

9.77.2.24 ga_mutat_adaptive()

procedure, public the_population::population::ga_mutat_adaptive
This function implements adaptive mutation rate that increases as the population size reduces. See the_population::population_ga_mutation_rate_adaptive().
Definition at line 162 of file m_popul.f90.

9.77.2.25 lifecycle_step()

procedure, public the_population::population::lifecycle_step
Perform a single step of the life cycle of the population. See the_population::population_lifecycle_step_preevol
Definition at line 166 of file m_popul.f90.

9.77.2.26 lifecycle_eatonly()

procedure, public the_population::population::lifecycle_eatonly
Perform a single step of the life cycle of the population. This version includes only optimal food selection and eating without the full fledged behaviour selection cascade of procedures ::do_behave(). See the_population::population_lifecycle_step_eatonly_preevol().
Definition at line 172 of file m_popul.f90.

9.77.3 Member Data Documentation

9.77.3.1 population_size

integer the_population::population::population_size
The size of the population.
Definition at line 60 of file m_popul.f90.

9.77.3.2 individual

type(member_population), dimension(:), allocatable the_population::population::individual
POPULATION is represented by an array of objects of the type MEMBER_POPULATION
POPULATION is an array of objects of the type MEMBER_POPULATION. It is represented as the %individual component of the type. The i-th individual of the population this is accessed as thisindividual(i). the population also has two descriptors: integer pop_number and string pop_name.
Definition at line 68 of file m_popul.f90.

9.77.3.3 pop_number

integer the_population::population::pop_number
The numeric ID of the population (if we have several populations))
Definition at line 70 of file m_popul.f90.

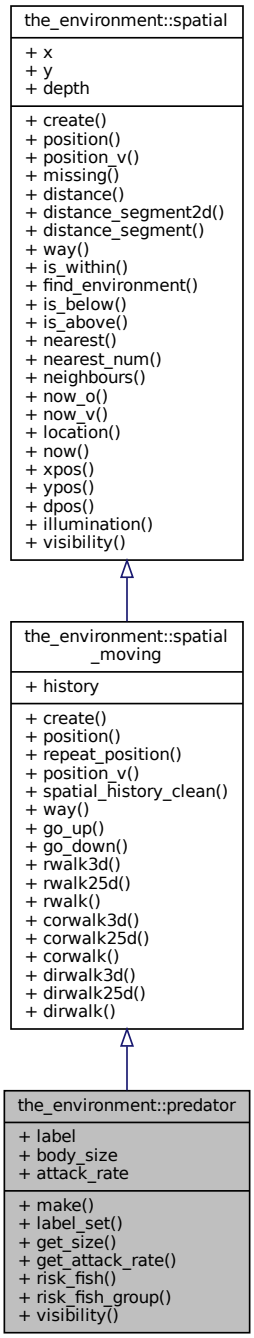
9.77.3.4 pop_name

character (len=label_length) the_population::population::pop_name
The descriptive name of the population.
Definition at line 72 of file m_popul.f90.
The documentation for this type was generated from the following file:

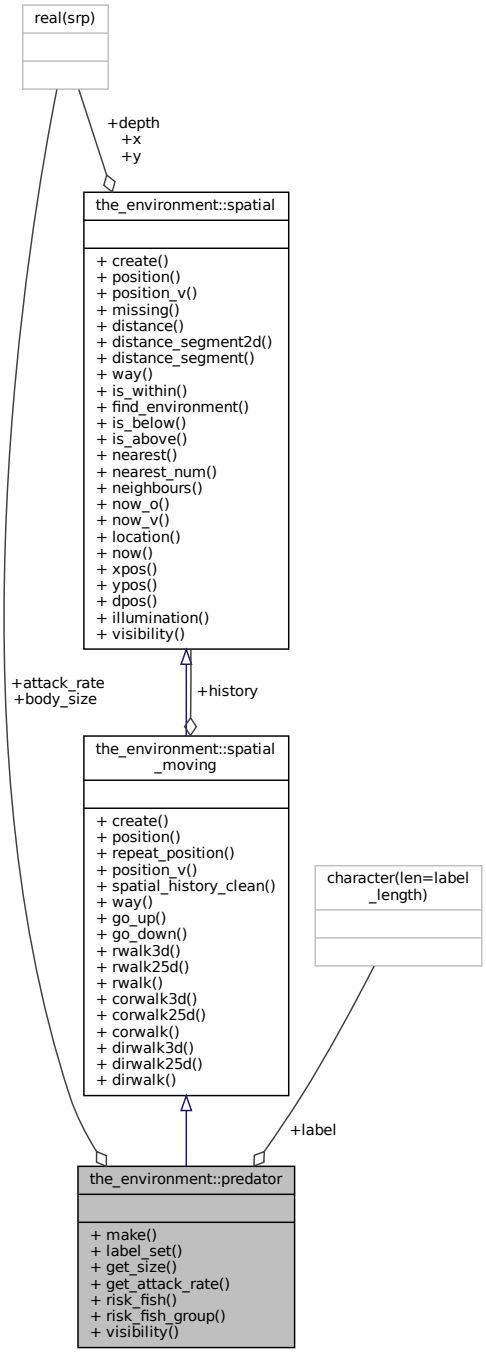
- m_popul.f90

9.78 the_environment::predator Type Reference

Definition of the PREDATOR objects. **Predator** is a moving agent that hunts on the evolving AHA agents but its internal structure is very simplistic (although we can in principle doit as a full AHA complexity with genome, GOS etc...).
Inheritance diagram for the_environment::predator:



Collaboration diagram for the_environment::predator:



Public Member Functions

- procedure, public make => predator_make_init

  Initialise a predator object. See `the_environment::predator_make_init()`
- procedure, public label_set => predator_label_set

  Set label for the predator, if not provided, set it random. See `the_environment::predator_label_set()`
- procedure, public get_size => predator_get_body_size

  Accessor function for the predator body size (length). See `the_environment::predator_get_body_size()`

- procedure, public get_attack_rate => predator_get_attack_rate

  Accessor function for the predator attack rate . See `the_environment::predator_get_capture_efficiency()`

- procedure, public risk_fish => predator_capture_risk_calculate_fish

  Calculates the risk of capture of the `prey_spatial` *idealised spatial object with the body length* `prey_length`*. This is a backend function. See* `the_environment::predator_capture_risk_calculate_fish()`*.*

- procedure, public risk_fish_group => predator_capture_risk_calculate_fish_group

  Calculates the risk of capture by a specific predator of an array of the fish agents with the spatial locations array defined by `prey_spatial` *and the body length array* `prey_length`*. This subroutine takes account of both the predator dilution and confusion effects and risk adjusted by the distance towards the predator. See* `the_environment::predator_capture_risk_calculate_fish_group()`*.*

- procedure, public visibility => predator_visibility_visual_range

  Calculate the visibility range of this predator. Wrapper to the `visual_range` *function. This function calculates the distance from which this predator can be seen by a visual object (e.g. prey). See* `the_environment::predator_visibility_visual_range()`*.*

Public Attributes

- character(len=label_length) label

  The label of the predator.

- real(srp) body_size

  Individual body size of the predator, can be stochastic or not. Can affect attack rate (e.g. larger predators more dangerous).

- real(srp) attack_rate

  The attack rate of the predator, i.e. the baseline probability of attacking catching the prey agent if the latter is found in proximity (within the visual range).

9.78.1 Detailed Description

Definition of the `PREDATOR` objects. **Predator** is a moving agent that hunts on the evolving AHA agents but its internal structure is very simplistic (although we can in principle doit as a full AHA complexity with genome, GOS etc...).
Definition at line 506 of file m_env.f90.

9.78.2 Member Function/Subroutine Documentation

9.78.2.1 make()

`procedure, public the_environment::predator::make`
Initialise a predator object. See `the_environment::predator_make_init()`
Definition at line 519 of file m_env.f90.

9.78.2.2 label_set()

`procedure, public the_environment::predator::label_set`
Set label for the predator, if not provided, set it random. See `the_environment::predator_label_set()`
Definition at line 522 of file m_env.f90.

9.78.2.3 get_size()

`procedure, public the_environment::predator::get_size`
Accessor function for the predator body size (length). See `the_environment::predator_get_body_size()`
Definition at line 525 of file m_env.f90.

9.78.2.4 get_attack_rate()

procedure, public the_environment::predator::get_attack_rate
Accessor function for the predator attack rate . See the_environment::predator_get_capture_↩efficiency()
Definition at line 528 of file m_env.f90.

9.78.2.5 risk_fish()

procedure, public the_environment::predator::risk_fish
Calculates the risk of capture of the prey_spatial idealised spatial object with the body length prey_length. This is a backend function. See the_environment::predator_capture_risk_calculate_fish().
Definition at line 532 of file m_env.f90.

9.78.2.6 risk_fish_group()

procedure, public the_environment::predator::risk_fish_group
Calculates the risk of capture by a specific predator of an array of the fish agents with the spatial locations array defined by prey_spatial and the body length array prey_length. This subroutine takes account of both the predator dilution and confusion effects and risk adjusted by the distance towards the predator. See the_environment::predator_capture_risk_calculate_fish_group().
Definition at line 540 of file m_env.f90.

9.78.2.7 visibility()

procedure, public the_environment::predator::visibility
Calculate the visibility range of this predator. Wrapper to the visual_range function. This function calculates the distance from which this predator can be seen by a visual object (e.g. prey). See the_environment::predator_visibility_visual_range().
Definition at line 546 of file m_env.f90.

9.78.3 Member Data Documentation

9.78.3.1 label

character (len=label_length) the_environment::predator::label
The label of the predator.
Definition at line 508 of file m_env.f90.

9.78.3.2 body_size

real(srp) the_environment::predator::body_size
Individual body size of the predator, can be stochastic or not. Can affect attack rate (e.g. larger predators more dangerous).
Definition at line 511 of file m_env.f90.

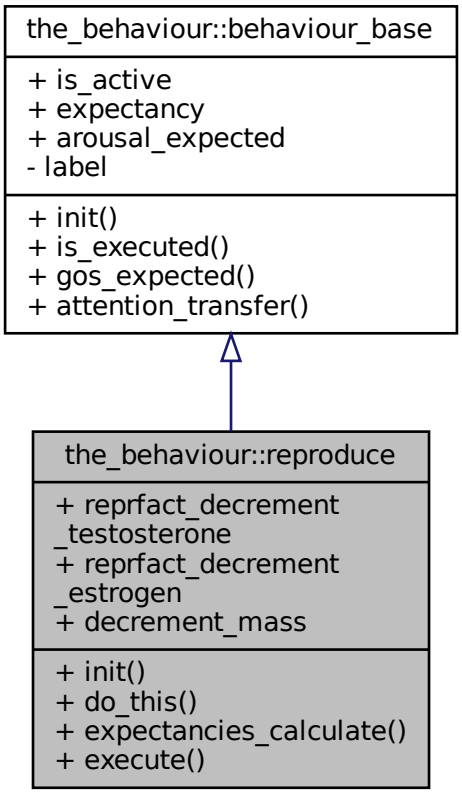
9.78.3.3 attack_rate

real(srp) the_environment::predator::attack_rate
The attack rate of the predator, i.e. the baseline probability of attacking catching the prey agent if the latter is found in proximity (within the visual range).
Definition at line 515 of file m_env.f90.
The documentation for this type was generated from the following file:

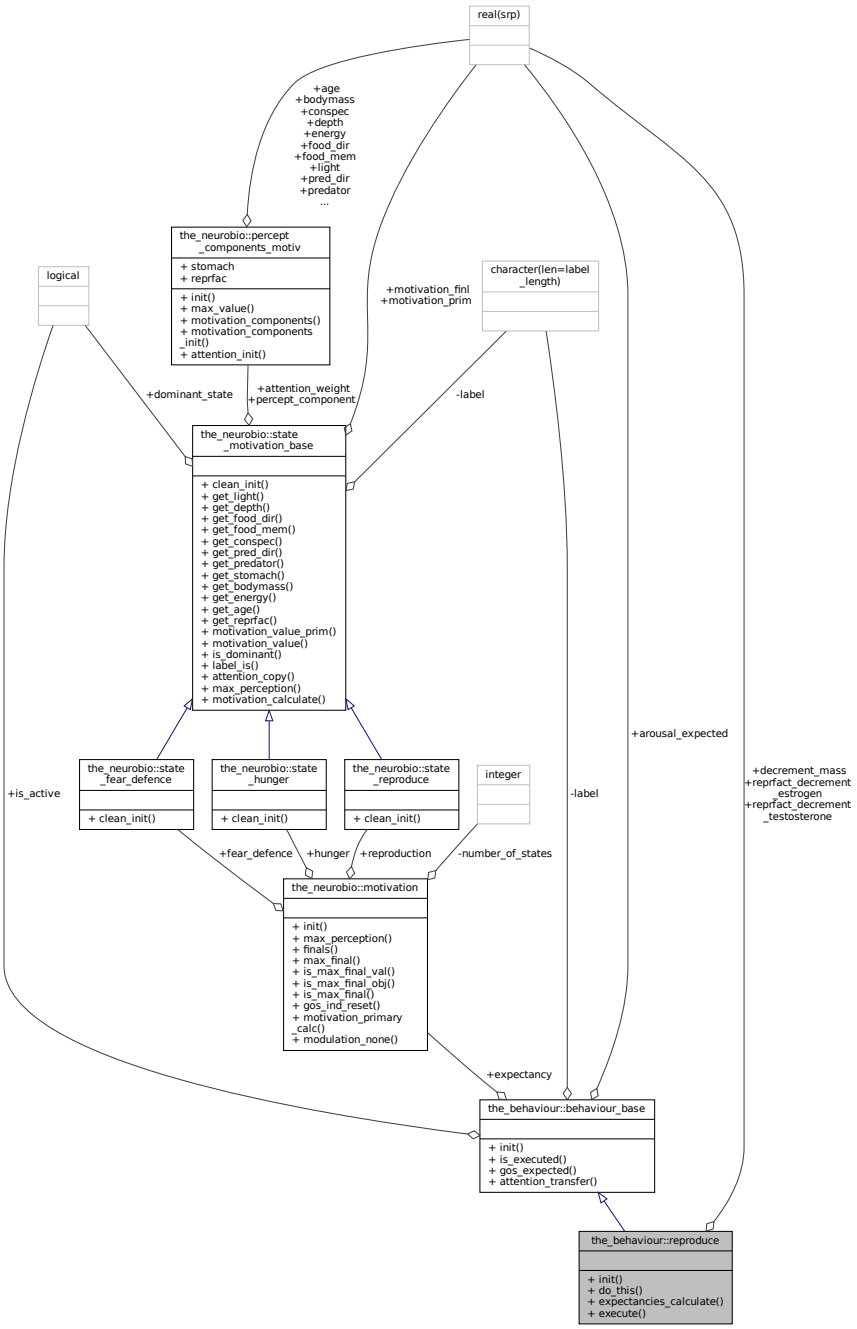
- m_env.f90

9.79 the_behaviour::reproduce Type Reference

Reproduce is do a single reproduction.
Inheritance diagram for the_behaviour::reproduce:



Collaboration diagram for the_behaviour::reproduce:



Public Member Functions

- procedure, public init => reproduce_init_zero

  Initialise reproduce behaviour object. See `the_behaviour::reproduce_init_zero()`.
- procedure, public do_this => reproduce_do_this

  Do reproduce by `this_agent` *(the actor agent) given the specific probability of successful reproduction. See* `the_behaviour::reproduce_do_this()`.
- procedure, public expectancies_calculate => reproduce_motivations_expect

  `expectancies_calculate` *is a subroutine (re)calculating motivations from fake expected perceptions following*

from `reproduce::do_this()` => `the_behaviour::reproduce_do_this()` *procedure. See implementation in* `the_behaviour::reproduce_motivations_expect()`.

- procedure, public execute => reproduce_do_execute

  Execute this behaviour component "reproduce" by the `this_agent` *agent. See* `the_behaviour::reproduce_do_execute()`

Public Attributes

- real(srp) reprfact_decrement_testosterone

  Decrement of the agent's current reproductive factor (sex-specific sex steroids level): testosterone.

- real(srp) reprfact_decrement_estrogen

  Decrement of the agent's current reproductive factor (sex-specific sex steroids level): estrogen.

- real(srp) decrement_mass

  Decrement of the agent's body mass resulting from this reproduction event object. The objective value is calculated via the `appraisal::probability_reproduction()` *method.*

9.79.1 Detailed Description

Reproduce is do a single reproduction.
Definition at line 144 of file m_behav.f90.

9.79.2 Member Function/Subroutine Documentation

9.79.2.1 init()

`procedure, public the_behaviour::reproduce::init`
Initialise reproduce behaviour object. See `the_behaviour::reproduce_init_zero()`.
Definition at line 158 of file m_behav.f90.

9.79.2.2 do_this()

`procedure, public the_behaviour::reproduce::do_this`
Do reproduce by `this_agent` (the actor agent) given the specific probability of successful reproduction. See `the_behaviour::reproduce_do_this()`.
Definition at line 162 of file m_behav.f90.

9.79.2.3 expectancies_calculate()

`procedure, public the_behaviour::reproduce::expectancies_calculate`
`expectancies_calculate` is a subroutine (re)calculating motivations from fake expected perceptions following from `reproduce::do_this()` => `the_behaviour::reproduce_do_this()` procedure. See implementation in `the_behaviour::reproduce_motivations_expect()`.
Definition at line 168 of file m_behav.f90.

9.79.2.4 execute()

`procedure, public the_behaviour::reproduce::execute`
Execute this behaviour component "reproduce" by the `this_agent` agent. See `the_behaviour::reproduce_do_execute`
Definition at line 172 of file m_behav.f90.

9.79.3 Member Data Documentation

9.79.3.1 reprfact_decrement_testosterone

`real(srp) the_behaviour::reproduce::reprfact_decrement_testosterone`
Decrement of the agent's current reproductive factor (sex-specific sex steroids level): testosterone.
Definition at line 147 of file m_behav.f90.

9.79.3.2 reprfact_decrement_estrogen

`real(srp) the_behaviour::reproduce::reprfact_decrement_estrogen`
Decrement of the agent's current reproductive factor (sex-specific sex steroids level): estrogen.
Definition at line 150 of file m_behav.f90.

9.79.3.3 decrement_mass

`real(srp) the_behaviour::reproduce::decrement_mass`
Decrement of the agent's body mass resulting from this reproduction event object. The objective value is calculated via the `appraisal::probability_reproduction()` method.
Definition at line 154 of file m_behav.f90.
The documentation for this type was generated from the following file:

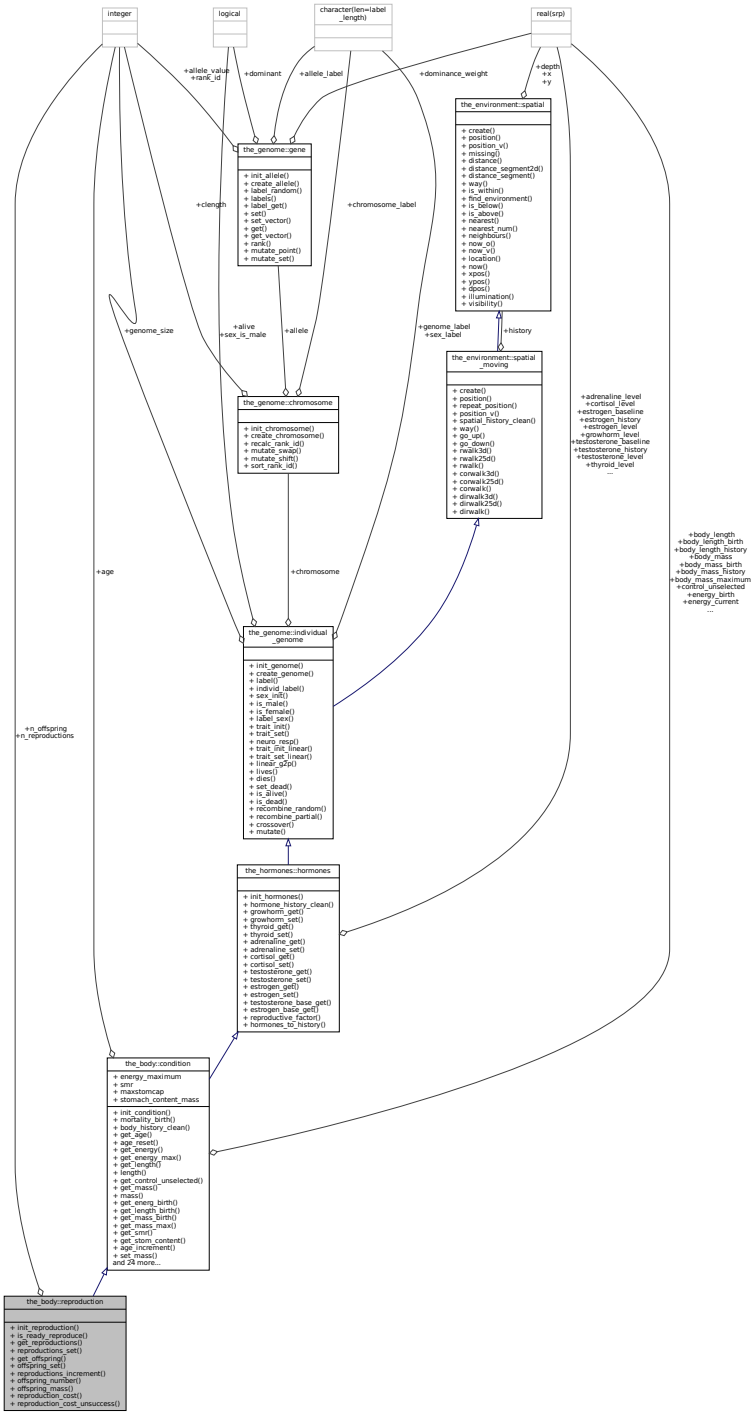
- m_behav.f90

9.80 the_body::reproduction Type Reference

`REPRODUCTION` type defines parameters of the reproduction system.

Inheritance diagram for the_body::reproduction:

Collaboration diagram for the_body::reproduction:



Public Member Functions

- procedure, public init_reproduction => reproduction_init_zero

  Init reproduction class. See `the_body::reproduction_init_zero()`.
- procedure, public is_ready_reproduce => reproduction_ready_steroid_hormones_exceed

  Determine if the agent's hormonal system is ready for reproduction See `the_body::reproduction_ready_steroid_hormone`
- procedure, public get_reproductions => reproduction_n_reproductions_get

  Get the number of reproductions for this agent. See `the_body::reproduction_n_reproductions_get()`.

- procedure, public reproductions_set => reproduction_n_reproductions_set

  Set the number of reproductions for the agent. See `the_body::reproduction_n_reproductions_set()`.
- procedure, public get_offspring => reproduction_n_offspring_get

  Get the number of offspring for this agent for its lifespan. See `the_body::reproduction_n_offspring_get()`.
- procedure, public offspring_set => reproduction_n_offspring_set

  Set the number of offspring this agent had during its lifespan. See `the_body::reproduction_n_offspring_set()`.
- procedure, public reproductions_increment => reproduction_n_increment

  Increment the number of reproductions and offspring for the agent. See `the_body::reproduction_n_increment()`.
- procedure, public offspring_number => reproduction_n_offspring_calc

  Calculate the number of offspring per a single reproduction. See `the_body::reproduction_n_offspring_calc()`.
- procedure, public offspring_mass => reproduction_mass_offspring_calc

  Calculate the total mass of all offspring per single reproduction. See `the_body::reproduction_mass_offspring_calc()`.
- procedure, public reproduction_cost => reproduction_cost_energy_dynamic

  Calculate the energetic cost of reproduction.
- procedure, public reproduction_cost_unsuccess => reproduction_cost_unsuccessful_calc

  Calculate the costs of unsuccessful reproduction. This is calculated as a fraction of the normal cost of reproduction returned by the function `reproduction::reproduction_cost()`. *See* `the_body::reproduction_cost_unsuccessfu`

Public Attributes

- integer n_reproductions

  Total number of reproductions during the lifespan.
- integer n_offspring

  Total number of offspring reproduced during the lifespan.

9.80.1 Detailed Description

`REPRODUCTION` type defines parameters of the reproduction system.
Definition at line 252 of file m_body.f90.

9.80.2 Member Function/Subroutine Documentation

9.80.2.1 init_reproduction()

`procedure, public the_body::reproduction::init_reproduction`
Init reproduction class. See `the_body::reproduction_init_zero()`.
Definition at line 260 of file m_body.f90.

9.80.2.2 is_ready_reproduce()

`procedure, public the_body::reproduction::is_ready_reproduce`
Determine if the agent's hormonal system is ready for reproduction See `the_body::reproduction_ready_steroid_horm`
Definition at line 264 of file m_body.f90.

9.80.2.3 get_reproductions()

`procedure, public the_body::reproduction::get_reproductions`
Get the number of reproductions for this agent. See `the_body::reproduction_n_reproductions_get()`.
Definition at line 269 of file m_body.f90.

9.80.2.4 reproductions_set()

`procedure, public the_body::reproduction::reproductions_set`
Set the number of reproductions for the agent. See `the_body::reproduction_n_reproductions_set()`.
Definition at line 272 of file m_body.f90.

9.80.2.5 get_offspring()

`procedure, public the_body::reproduction::get_offspring`
Get the number of offspring for this agent for its lifespan. See `the_body::reproduction_n_offspring_get()`.
Definition at line 275 of file m_body.f90.

9.80.2.6 offspring_set()

`procedure, public the_body::reproduction::offspring_set`
Set the number of offspring this agent had during its lifespan. See `the_body::reproduction_n_offspring_set()`.
Definition at line 278 of file m_body.f90.

9.80.2.7 reproductions_increment()

`procedure, public the_body::reproduction::reproductions_increment`
Increment the number of reproductions and offspring for the agent. See `the_body::reproduction_n_increment()`.
Definition at line 281 of file m_body.f90.

9.80.2.8 offspring_number()

`procedure, public the_body::reproduction::offspring_number`
Calculate the number of offspring per a single reproduction. See `the_body::reproduction_n_offspring_calc()`.
Definition at line 284 of file m_body.f90.

9.80.2.9 offspring_mass()

`procedure, public the_body::reproduction::offspring_mass`
Calculate the total mass of all offspring per single reproduction. See `the_body::reproduction_mass_offspring_calc(`
Definition at line 287 of file m_body.f90.

9.80.2.10 reproduction_cost()

`procedure, public the_body::reproduction::reproduction_cost`
Calculate the energetic cost of reproduction.

Note

Two versions are implemented:

- `the_body::reproduction_cost_energy_fix()`
- `the_body::reproduction_cost_energy_dynamic()`

Definition at line 293 of file m_body.f90.

9.80.2.11 reproduction_cost_unsuccess()

`procedure, public the_body::reproduction::reproduction_cost_unsuccess`
Calculate the costs of unsuccessful reproduction. This is calculated as a fraction of the normal cost of reproduction returned by the function `reproduction::reproduction_cost()`. See `the_body::reproduction_cost_unsucce`
Definition at line 298 of file m_body.f90.

9.80.3 Member Data Documentation

9.80.3.1 n_reproductions

`integer the_body::reproduction::n_reproductions`
Total number of reproductions during the lifespan.
Definition at line 254 of file m_body.f90.

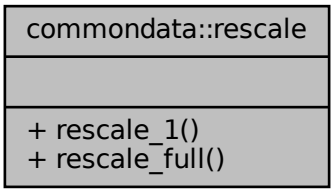
9.80.3.2 n_offspring

`integer the_body::reproduction::n_offspring`
Total number of offspring reproduced during the lifespan.
Definition at line 256 of file m_body.f90.
The documentation for this type was generated from the following file:

- m_body.f90

9.81 commondata::rescale Interface Reference

Arbitrary rescales value(s) from one range (A:B) to another (A1:B1).
Collaboration diagram for commondata::rescale:



Public Member Functions

- elemental real(srp) function rescale_1 (value_in, A1, B1)

  Rescale a real variable with the range 0:1 to have the new range A1:B1.
- elemental real(srp) function rescale_full (value_in, A, B, A1, B1)

  Rescale a real variable with the range A:B to have the new range A1:B1.

9.81.1 Detailed Description

Arbitrary rescales value(s) from one range (A:B) to another (A1:B1).
Rescales values from A:B to A1:B1, or (if only A1:B1 are provided) from 0:1 to A1:B1.

Note

Elemental functions.

Definition at line 5341 of file m_common.f90.

9.81.2 Member Function/Subroutine Documentation

9.81.2.1 rescale_1()

```
elemental real(srp) function commondata::rescale::rescale_1 (
             real(srp), intent(in) value_in,
             real(srp), intent(in) A1,
             real(srp), intent(in) B1 )
```

Rescale a real variable with the range 0:1 to have the new range A1:B1.

Warning

The function does not check if `value_in` lies within [0:1].

Note

Code for wxMaxima equation solve:
`solve( [a1=0*k+b, b1=1*k+b] ,[k,b] );`

Definition at line 5734 of file m_common.f90.

9.81.2.2 rescale_full()

```
elemental real(srp) function commondata::rescale::rescale_full (
             real(srp), intent(in) value_in,
             real(srp), intent(in) A,
             real(srp), intent(in) B,
             real(srp), intent(in) A1,
             real(srp), intent(in) B1 )
```

Rescale a real variable with the range A:B to have the new range A1:B1.
Linear transformation of the input value `value_in` such `k * value_in + beta`, where the `k` and `beta` coefficients are found by solving a simple linear system: $\{ A_1 = k \cdot A + \beta; B_1 = k \cdot B + \beta$. It has this solution: $k = \frac{A_1 - B_1}{A - B}, \beta = -\frac{A_1 \cdot B - A \cdot B_1}{A - B}$

Warning

The function does not check if `value_in` lies within [A:B].

Note

Code for wxMaxima equation solve:
`solve( [a1=a*k+b, b1=b*k+b] ,[k,b] );`

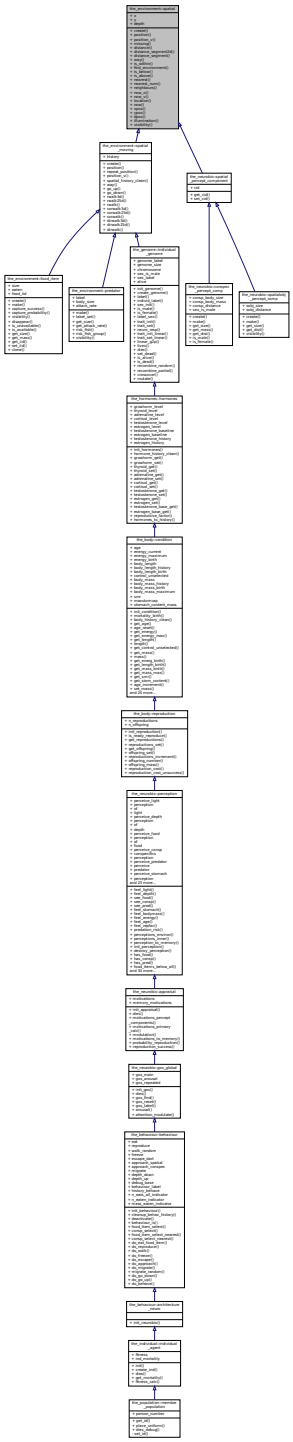
Definition at line 5706 of file m_common.f90.
The documentation for this interface was generated from the following file:

- m_common.f90

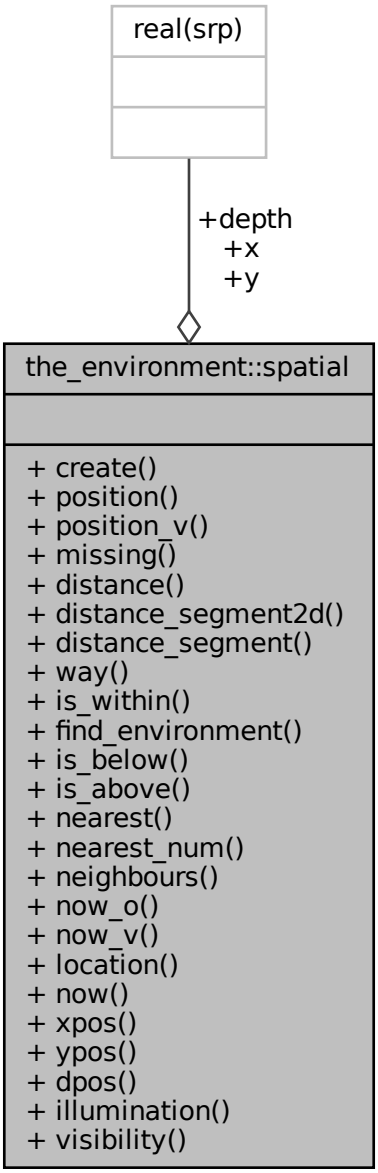
9.82 the_environment::spatial Type Reference

Definition of a spatial object. Spatial object determines the position of the agent, food items and other things in the simulated space. Here we use continuous 3D environment (real type coordinates)

Inheritance diagram for the_environment::spatial:



Collaboration diagram for the_environment::spatial:



Public Member Functions

- procedure, public create => spatial_create_empty

  Create an empty spatial object.
- procedure, public position => spatial_fix_position_3d_o

  Place spatial object into a 3D space, define the object's current coordinates. Object-based procedure. See `the_environment::spatial_fix_position_3d_o()`
- procedure, public position_v => spatial_fix_position_3d_s

  Place spatial object into a 3D space, define the object's current coordinates. Vector-based procedure. See `the_environment::spatial_fix_position_3d_s()`

- procedure, public missing => spatial_make_missing

  Assign all MISSING coordinates to the SPATIAL object. See the_environment::spatial_make_missing()
- procedure, public distance => spatial_distance_3d

  Calculate the Euclidean distance between two spatial objects. See the_environment::spatial_distance_3d()
- procedure, public distance_segment2d => geo_poly2d_dist_point_to_section

  Calculates the minimum distance from a the_environment::spatial class object to a line segment delimited by two the_environment::spatial endpoints in the 2D XY plane (the depth coordinate is ignored). See the_environment::geo_poly2d_dist_point_to_section().
- procedure, public distance_segment => geo_poly3d_dist_point_to_section

  Calculates the minimum distance from a the_environment::spatial class object to a line segment delimited by two the_environment::spatial class endpoints in the 3D XY space. See the_environment::geo_poly3d_dist_point_to_secti
- procedure, public way => spatial_self_distance_3d

  Calculate the Euclidean distance between the current and previous position of a single spatial object. See the_environment::spatial_self_distance_3d()
- procedure, public is_within => spatial_check_located_within_3d

  Function to check if this spatial object is located within an area set by an environmental object See the_environment::spatial_check_located_within_3d()
- procedure, public find_environment => spatial_get_environment_in_pos

  Identify in which environment from the input list this spatial agent is currently in. See the_environment::spatial_get_enviro
- procedure, public is_below => spatial_check_located_below

  *Logical function to check if the argument spatial object(s) is(are) located **below** this spatial object. See the_environment::spatial_check_located_below()*
- procedure, public is_above => spatial_check_located_above

  *Logical function to check if the argument spatial object(s) is(are) located **above** this spatial object. See the_environment::spatial_check_located_above()*
- procedure, public nearest => spatial_get_nearest_object

  *Determine the nearest spatial object to **this** spatial object among an array of other spatial objects. See the_environment::spatial_get_nearest_object()*
- procedure, public nearest_num => spatial_get_nearest_id

  *Determine the nearest spatial object to **this** spatial object among an array of other spatial objects. See the_environment::spatial_get_nearest_id()*
- procedure, public neighbours => spatial_neighbours_distances

  *Calculate the distances between **this** spatial object and an array of its neighbours. Optionally output the distances, sorting index vector and rankings vector for each of these neighbours. Optionally do only partial indexing, up to the order rank_max for computational speed. See the_environment::spatial_neighbours_distances()*
- procedure, public now_o => spatial_get_current_pos_3d_o

  Get the current spatial position of a SPATIAL object. Object-based. See the_environment::spatial_get_current_pos_3
- procedure, public now_v => spatial_get_current_pos_3d_v

  Get the current spatial position of a SPATIAL object. Vector-based. See the_environment::spatial_get_current_pos_3
- generic, public location => now_o, now_v

  Get the current spatial position of a SPATIAL object. Generic interface/alias.
- generic, public now => now_o, now_v

  Get the current spatial position of a SPATIAL object. Generic interface/alias.
- procedure, public xpos => spatial_get_current_pos_x_3d

  Get the current X position of a SPATIAL object. See the_environment::spatial_get_current_pos_x_3d()
- procedure, public ypos => spatial_get_current_pos_y_3d

  Get the current Y position of a SPATIAL object. See the_environment::spatial_get_current_pos_y_3d()
- procedure, public dpos => spatial_get_current_pos_d_3d

  Get the current Z (depth) position of a SPATIAL object. See the_environment::spatial_get_current_pos_d_3d()
- procedure, public illumination => spatial_calc_irradiance_at_depth

  Calculate the illumination (background irradiance) at the depth of the spatial object at an arbitrary time step of the model. See the_environment::spatial_calc_irradiance_at_depth()
- procedure, public visibility => spatial_visibility_visual_range_cm

  Calculate the visibility range of a spatial object. Wrapper to the visual_range function. This function calculates the distance from which this object can be seen by a visual object (e.g. predator or prey). See the_environment::spatial_visibility_visual_range_cm().

Public Attributes

- real(srp) x

  We define three-dimensional environment: x, y and depth.
- real(srp) y
- real(srp) depth

9.82.1 Detailed Description

Definition of a spatial object. Spatial object determines the position of the agent, food items and other things in the simulated space. Here we use continuous 3D environment (real type coordinates)

Note

IMPORTANT: We will use **position** method to **set** location of a spatial object and **location** method to **get** its location.

We use `position_v` method to **set** define spatial position using raw 3D coordinates, x, y, z – it lacks extensibility – for convenience only.

Warning

Note that we **do not** set **ID** at the elementary `SPATIAL` objects. This is done to make type constructor `SPATIAL(x, y, z)` (that is used frequently in different places) shorter and easier to use. We do not need to include an ID then. IDs are set at higher levels in the object hierarchy, e.g. `FOOD_ITEM` has `food_iid` integer data component.

Definition at line 50 of file m_env.f90.

9.82.2 Member Function/Subroutine Documentation

9.82.2.1 create()

`procedure, public the_environment::spatial::create`
Create an empty spatial object.

Note

`position` is the standard method for placing spatial object, returns object type. `position_v` is non-extensible method that returns raw 3D coordinates, for convenience only. `position` and `position_v` are overriden in moving objects below. Not sure if we really need `position_v`. See `the_environment::spatial_create_empty()`

Definition at line 61 of file m_env.f90.

9.82.2.2 position()

`procedure, public the_environment::spatial::position`
Place spatial object into a 3D space, define the object's current coordinates. Object-based procedure. See `the_environment::spatial_fix_position_3d_o()`
Definition at line 65 of file m_env.f90.

9.82.2.3 position_v()

`procedure, public the_environment::spatial::position_v`
Place spatial object into a 3D space, define the object's current coordinates. Vector-based procedure. See `the_environment::spatial_fix_position_3d_s()`
Definition at line 69 of file m_env.f90.

9.82.2.4 missing()

procedure, public the_environment::spatial::missing
Assign all MISSING coordinates to the SPATIAL object. See the_environment::spatial_make_missing()
Definition at line 72 of file m_env.f90.

9.82.2.5 distance()

procedure, public the_environment::spatial::distance
Calculate the Euclidean distance between two spatial objects. See the_environment::spatial_distance_3d()
Definition at line 75 of file m_env.f90.

9.82.2.6 distance_segment2d()

procedure, public the_environment::spatial::distance_segment2d
Calculates the minimum distance from a the_environment::spatial class object to a line segment delimited by two the_environment::spatial endpoints in the 2D *XY* plane (the depth coordinate is ignored). See the_environment::geo_poly2d_dist_point_to_section().
Definition at line 80 of file m_env.f90.

9.82.2.7 distance_segment()

procedure, public the_environment::spatial::distance_segment
Calculates the minimum distance from a the_environment::spatial class object to a line segment delimited by two the_environment::spatial class endpoints in the 3D *XY* space. See the_environment::geo_poly3d_dist_point_to_se
Definition at line 86 of file m_env.f90.

9.82.2.8 way()

procedure, public the_environment::spatial::way
Calculate the Euclidean distance between the current and previous position of a single spatial object. See the_environment::spatial_self_distance_3d()
Definition at line 91 of file m_env.f90.

9.82.2.9 is_within()

procedure, public the_environment::spatial::is_within
Function to check if this spatial object is located within an area set by an environmental object See the_environment::spatial_check_located_within_3d()
Definition at line 95 of file m_env.f90.

9.82.2.10 find_environment()

procedure, public the_environment::spatial::find_environment
Identify in which environment from the input list this spatial agent is currently in. See the_environment::spatial_get_envi
Definition at line 99 of file m_env.f90.

9.82.2.11 is_below()

procedure, public the_environment::spatial::is_below
Logical function to check if the argument spatial object(s) is(are) located **below** this spatial object. See the_environment::spatial_check_located_below()
Definition at line 103 of file m_env.f90.

9.82.2.12 is_above()

procedure, public the_environment::spatial::is_above
Logical function to check if the argument spatial object(s) is(are) located **above** this spatial object. See the_environment::spatial_check_located_above()
Definition at line 107 of file m_env.f90.

9.82.2.13 nearest()

procedure, public the_environment::spatial::nearest
Determine the nearest spatial object to **this** spatial object among an array of other spatial objects. See the_environment::spatial_get_nearest_object()
Definition at line 111 of file m_env.f90.

9.82.2.14 nearest_num()

procedure, public the_environment::spatial::nearest_num
Determine the nearest spatial object to **this** spatial object among an array of other spatial objects. See the_environment::spatial_get_nearest_id()
Definition at line 115 of file m_env.f90.

9.82.2.15 neighbours()

procedure, public the_environment::spatial::neighbours
Calculate the distances between **this** spatial object and an array of its neighbours. Optionally output the distances, sorting index vector and rankings vector for each of these neighbours. Optionally do only partial indexing, up to the order rank_max for computational speed. See the_environment::spatial_neighbours_distances()
Definition at line 121 of file m_env.f90.

9.82.2.16 now_o()

procedure, public the_environment::spatial::now_o
Get the current spatial position of a SPATIAL object. Object-based. See the_environment::spatial_get_current_pos
Definition at line 124 of file m_env.f90.

9.82.2.17 now_v()

procedure, public the_environment::spatial::now_v
Get the current spatial position of a SPATIAL object. Vector-based. See the_environment::spatial_get_current_pos
Definition at line 127 of file m_env.f90.

9.82.2.18 location()

generic, public the_environment::spatial::location
Get the current spatial position of a SPATIAL object. Generic interface/alias.
Definition at line 130 of file m_env.f90.

9.82.2.19 now()

generic, public the_environment::spatial::now
Get the current spatial position of a SPATIAL object. Generic interface/alias.
Definition at line 133 of file m_env.f90.

9.82.2.20 xpos()

procedure, public the_environment::spatial::xpos
Get the current X position of a SPATIAL object. See the_environment::spatial_get_current_pos_x_3d()
Definition at line 136 of file m_env.f90.

9.82.2.21 ypos()

procedure, public the_environment::spatial::ypos
Get the current Y position of a SPATIAL object. See the_environment::spatial_get_current_pos_y_3d()
Definition at line 139 of file m_env.f90.

9.82.2.22 dpos()

procedure, public the_environment::spatial::dpos
Get the current Z (depth) position of a SPATIAL object. See the_environment::spatial_get_current_pos_d_3d()
Definition at line 142 of file m_env.f90.

9.82.2.23 illumination()

procedure, public the_environment::spatial::illumination
Calculate the illumination (background irradiance) at the depth of the spatial object at an arbitrary time step of the model. See the_environment::spatial_calc_irradiance_at_depth()
Definition at line 146 of file m_env.f90.

9.82.2.24 visibility()

procedure, public the_environment::spatial::visibility
Calculate the visibility range of a spatial object. Wrapper to the visual_range function. This function calculates the distance from which this object can be seen by a visual object (e.g. predator or prey). See the_environment::spatial_visibility_visual_range_cm().

Warning

> The function interface for the basic spatial type the_environment::spatial is called visibility_↩ basic to distinguish it from similar visibility methods defined for several extension classes, such as the_environment::predator, the_environment::food_item and the_body::condition because this function is unrelated to them, otherwise it must have the same parameters as in the class extensions.

Definition at line 160 of file m_env.f90.

9.82.3 Member Data Documentation

9.82.3.1 x

real(srp) the_environment::spatial::x
We define three-dimensional environment: x, y and depth.
Definition at line 52 of file m_env.f90.

9.82.3.2 y

`real(srp) the_environment::spatial::y`
Definition at line 52 of file m_env.f90.

9.82.3.3 depth

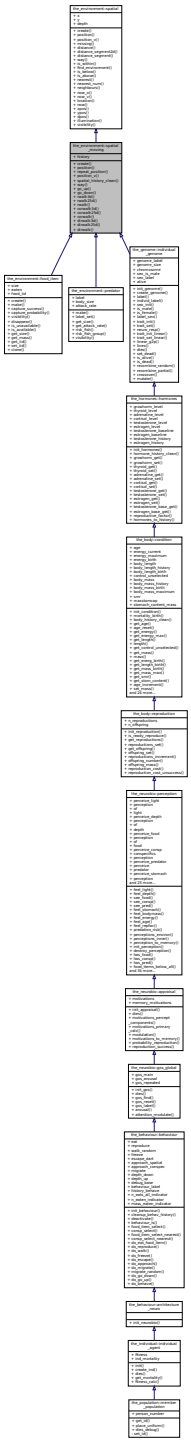
`real(srp) the_environment::spatial::depth`
Definition at line 52 of file m_env.f90.
The documentation for this type was generated from the following file:

- m_env.f90

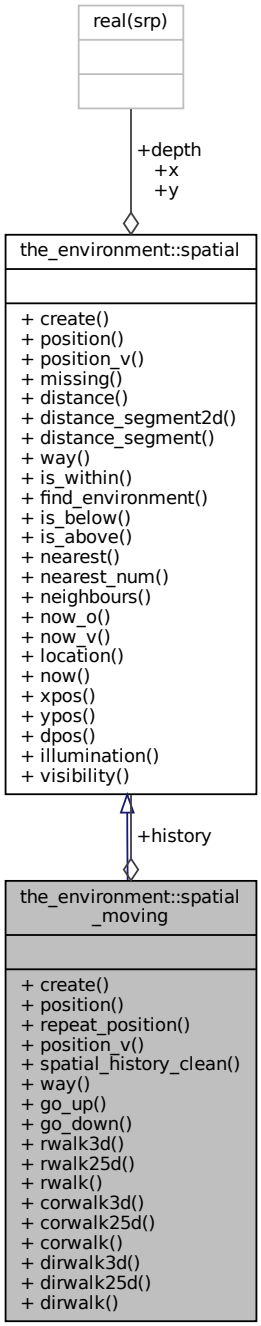
9.83 the_environment::spatial_moving Type Reference

Definition of a movable spatial object. It extends the the_environment::spatial object, but also adds its previous position history in stack-like arrays. The history is maintained with the commondata::add_to_history() subroutine, adding a single element on the top (end) of the history stack.

Inheritance diagram for the_environment::spatial_moving:



Collaboration diagram for the_environment::spatial_moving:



Public Member Functions

- procedure, public create => spatial_moving_create_3d

  Create a new spatial moving object. Initially it has no position, all coordinate values are commondata::missing or commondata::invalid for real type coordinates. See `the_environment::spatial_moving_create_3d()`

- procedure, public position => spatial_moving_fix_position_3d_o

  Place spatial movable object into a 3D space, define the object's current coordinates, but first save previous coordinates. Object-based. See `the_environment::spatial_moving_fix_position_3d_o()`

- procedure, public repeat_position => spatial_moving_repeat_position_history_3d

  Repeat/re-save the current position into the positional history stack. See `the_environment::spatial_moving_repeat_posi`
- procedure, public position_v => spatial_moving_fix_position_3d_v

  Place spatial movable object into a 3D space, define the object's current coordinates, but first save previous coordinates. Vector-based. See `the_environment::spatial_moving_fix_position_3d_v()`
- procedure, public spatial_history_clean => spatial_moving_clean_hstory_3d

  Create a new empty history of positions for spatial moving object. Assign all values to the commondata::missing value code. See `the_environment::spatial_moving_clean_hstory_3d()`
- procedure, public way => spatial_moving_self_distance_3d

  Calculate the Euclidean distance between the current and previous position of a single spatial movable object. Optionally, it also calculates the total distance traversed during the `from_history` *points from the history stack along with the distance from the current position and the last historical value. See* `the_environment::spatial_moving_self_distance_3d()`.
- procedure, public go_up => spatial_moving_go_up

  The spatial moving object ***ascends****, goes up the depth with specific fixed step size. See* `the_environment::spatial_moving_g`
- procedure, public go_down => spatial_moving_go_down

  The spatial moving object ***decends****, goes down the depth with specific fixed step size. See* `the_environment::spatial_moving`
- procedure, public rwalk3d => spatial_moving_randomwalk_gaussian_step_3d

  Implements an optionally environment-restricted Gaussian random walk in 3D. See `the_environment::spatial_moving_rand`
- procedure, public rwalk25d =>spatial_moving_randomwalk_gaussian_step_25d

  Implements an optionally environment-restricted Gaussian random walk in a "2.5 dimensions", i.e. 2D x y with separate walk parameters for the third depth dimension. See `the_environment::spatial_moving_randomwalk_gaussian_st`
- generic, public rwalk => rwalk3d, rwalk25d

  Implements an optionally environment-restricted Gaussian random walk. Generic interface for 3D and 3.↩ 5D moves. See `the_environment::spatial_moving_randomwalk_gaussian_step_3d()` *and* `the_environment::spatial_moving_randomwalk_gaussian_step_25d()`.
- procedure, public corwalk3d => spatial_moving_corwalk_gaussian_step_3d

  Implements an optionally environment-restricted ***correlated directional*** *Gaussian random walk in 3D towards (or away of) an the_environment::spatial class* `target` *object. See* `the_environment::spatial_moving_corwalk_gaussian`
- procedure, public corwalk25d =>spatial_moving_corwalk_gaussian_step_25d

  Implements an optionally environment-restricted ***correlated directional*** *Gaussian random walk in 3D towards (or away of) an the_environment::spatial class* `target` *object. See* `the_environment::spatial_moving_corwalk_gaussian`
- generic, public corwalk => corwalk3d, corwalk25d

  Implements an optionally environment-restricted ***correlated directional*** *Gaussian random walk.* `corwalk` *is a generic interface for 3D and "2.5"D moves. For details see the 3d version and a version with separate X,Y and depth random parameters.*
- procedure, public dirwalk3d => spatial_moving_dirwalk_gaussian_step_3d

  Implements an optionally environment-restricted ***directional*** *Gaussian random walk in 3D towards a* `target` *the_environment::spatial object. See* `the_environment::spatial_moving_dirwalk_gaussian_step_3d()`
- procedure, public dirwalk25d =>spatial_moving_dirwalk_gaussian_step_25d

  Implements an optionally environment-restricted ***directional*** *Gaussian random walk in "2.5"D towards a* `target` *object. i.e. 2D x y with separate walk parameters for the third depth dimension. See* `the_environment::spatial_moving_dirwalk_gaussian_step_25d()`
- generic, public dirwalk => dirwalk3d, dirwalk25d

  Implements an optionally environment-restricted ***directional*** *Gaussian random walk. Generic interface for 3D and "2.5"D moves.*

Public Attributes

- type(spatial), dimension(history_size_spatial) history

  We define prior historical values of the `SPATIAL` *positions.*

9.83.1 Detailed Description

Definition of a movable spatial object. It extends the the_environment::spatial object, but also adds its previous position history in stack-like arrays. The history is maintained with the commondata::add_to_history() subroutine, adding a single element on the top (end) of the history stack.
Definition at line 168 of file m_env.f90.

9.83.2 Member Function/Subroutine Documentation

9.83.2.1 create()

`procedure, public the_environment::spatial_moving::create`
Create a new spatial moving object. Initially it has no position, all coordinate values are commondata::missing or commondata::invalid for real type coordinates. See `the_environment::spatial_moving_create_3d()`
Definition at line 184 of file m_env.f90.

9.83.2.2 position()

`procedure, public the_environment::spatial_moving::position`
Place spatial movable object into a 3D space, define the object's current coordinates, but first save previous coordinates. Object-based. See `the_environment::spatial_moving_fix_position_3d_o()`
Definition at line 188 of file m_env.f90.

9.83.2.3 repeat_position()

`procedure, public the_environment::spatial_moving::repeat_position`
Repeat/re-save the current position into the positional history stack. See `the_environment::spatial_moving_repeat_po`
Definition at line 191 of file m_env.f90.

9.83.2.4 position_v()

`procedure, public the_environment::spatial_moving::position_v`
Place spatial movable object into a 3D space, define the object's current coordinates, but first save previous coordinates. Vector-based. See `the_environment::spatial_moving_fix_position_3d_v()`
Definition at line 196 of file m_env.f90.

9.83.2.5 spatial_history_clean()

`procedure, public the_environment::spatial_moving::spatial_history_clean`
Create a new empty history of positions for spatial moving object. Assign all values to the commondata::missing value code. See `the_environment::spatial_moving_clean_history_3d()`
Definition at line 200 of file m_env.f90.

9.83.2.6 way()

`procedure, public the_environment::spatial_moving::way`
Calculate the Euclidean distance between the current and previous position of a single spatial movable object. Optionally, it also calculates the total distance traversed during the `from_history` points from the history stack along with the distance from the current position and the last historical value. See `the_environment::spatial_moving_self_distance_3d()`.
Definition at line 207 of file m_env.f90.

9.83.2.7 go_up()

procedure, public the_environment::spatial_moving::go_up
The spatial moving object **ascends**, goes up the depth with specific fixed step size. See the_environment::spatial_moving
Definition at line 210 of file m_env.f90.

9.83.2.8 go_down()

procedure, public the_environment::spatial_moving::go_down
The spatial moving object **decends**, goes down the depth with specific fixed step size. See the_environment::spatial_movi
Definition at line 214 of file m_env.f90.

9.83.2.9 rwalk3d()

procedure, public the_environment::spatial_moving::rwalk3d
Implements an optionally environment-restricted Gaussian random walk in 3D. See the_environment::spatial_moving_ra
Definition at line 218 of file m_env.f90.

9.83.2.10 rwalk25d()

procedure, public the_environment::spatial_moving::rwalk25d
Implements an optionally environment-restricted Gaussian random walk in a "2.5 dimensions", i.e. 2D x y with separate walk parameters for the third depth dimension. See the_environment::spatial_moving_randomwalk_gaussian_
Definition at line 223 of file m_env.f90.

9.83.2.11 rwalk()

generic, public the_environment::spatial_moving::rwalk
Implements an optionally environment-restricted Gaussian random walk. Generic interface for 3D and 3.↩ 5D moves. See the_environment::spatial_moving_randomwalk_gaussian_step_3d() and the_environment::spatial_moving_randomwalk_gaussian_step_25d().
Definition at line 228 of file m_env.f90.

9.83.2.12 corwalk3d()

procedure, public the_environment::spatial_moving::corwalk3d
Implements an optionally environment-restricted **correlated directional** Gaussian random walk in 3D towards (or away of) an the_environment::spatial class target object. See the_environment::spatial_moving_corwalk_gauss
Definition at line 233 of file m_env.f90.

9.83.2.13 corwalk25d()

procedure, public the_environment::spatial_moving::corwalk25d
Implements an optionally environment-restricted **correlated directional** Gaussian random walk in 3D towards (or away of) an the_environment::spatial class target object. See the_environment::spatial_moving_corwalk_gauss
Definition at line 238 of file m_env.f90.

9.83.2.14 corwalk()

generic, public the_environment::spatial_moving::corwalk
Implements an optionally environment-restricted **correlated directional** Gaussian random walk. corwalk is a generic interface for 3D and "2.5"D moves. For details see the 3d version and a version with separate *X,Y* and *depth* random parameters.

- the_environment::spatial_moving_corwalk_gaussian_step_3d();
- the_environment::spatial_moving_corwalk_gaussian_step_25d();

Definition at line 246 of file m_env.f90.

9.83.2.15 dirwalk3d()

procedure, public the_environment::spatial_moving::dirwalk3d
Implements an optionally environment-restricted **directional** Gaussian random walk in 3D towards a target the_environment::spatial object. See the_environment::spatial_moving_dirwalk_gaussian_step_3d()

Warning

obsolete, will be removed!

Definition at line 252 of file m_env.f90.

9.83.2.16 dirwalk25d()

procedure, public the_environment::spatial_moving::dirwalk25d
Implements an optionally environment-restricted **directional** Gaussian random walk in "2.5"D towards a target object. i.e. 2D x y with separate walk parameters for the third depth dimension. See the_environment::spatial_moving_dirwalk_gaussian_step_25d()

Warning

obsolete, will be removed!

Definition at line 259 of file m_env.f90.

9.83.2.17 dirwalk()

generic, public the_environment::spatial_moving::dirwalk
Implements an optionally environment-restricted **directional** Gaussian random walk. Generic interface for 3D and "2.5"D moves.

Warning

obsolete, will be removed!

Definition at line 263 of file m_env.f90.

9.83.3 Member Data Documentation

9.83.3.1 history

type(spatial), dimension(history_size_spatial) the_environment::spatial_moving::history
We define prior historical values of the SPATIAL positions.

Note

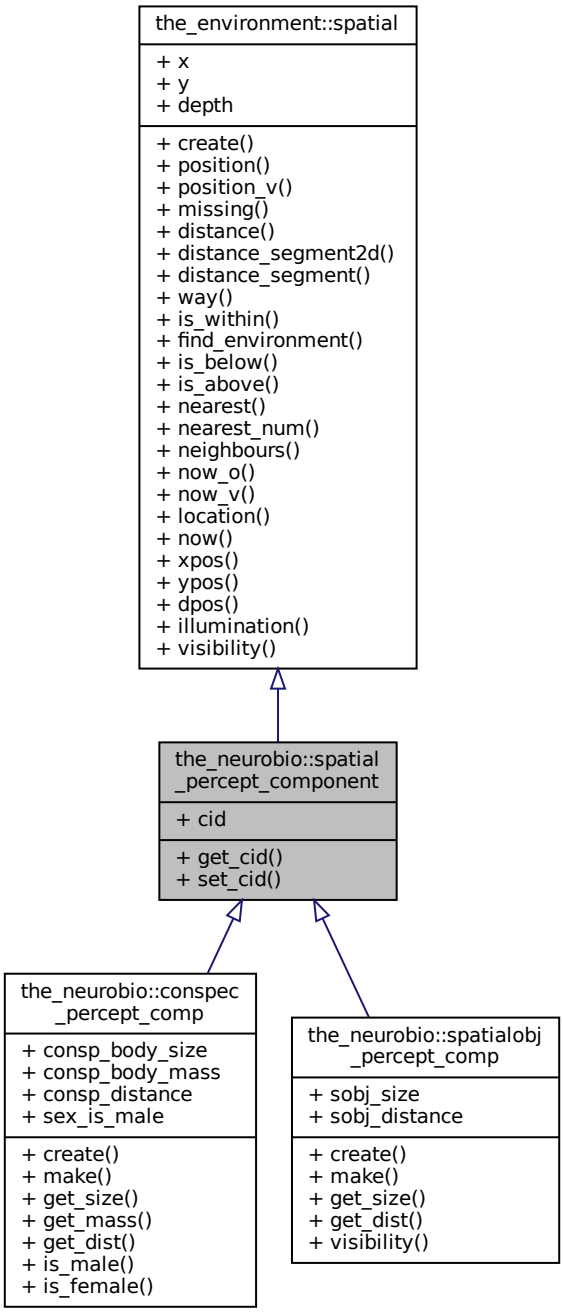
Historical stack has the same the_environment::spatial type but is an array of prior values (i.e. array of the_environment::spatial objects). The the_environment::spatial_moving::position() method overrides the standard function defined for the_environment::spatial (the_environment::spatial::position()), it not only sets the current position, but also moves the previous position of the object into the history stack.

Definition at line 178 of file m_env.f90.
The documentation for this type was generated from the following file:

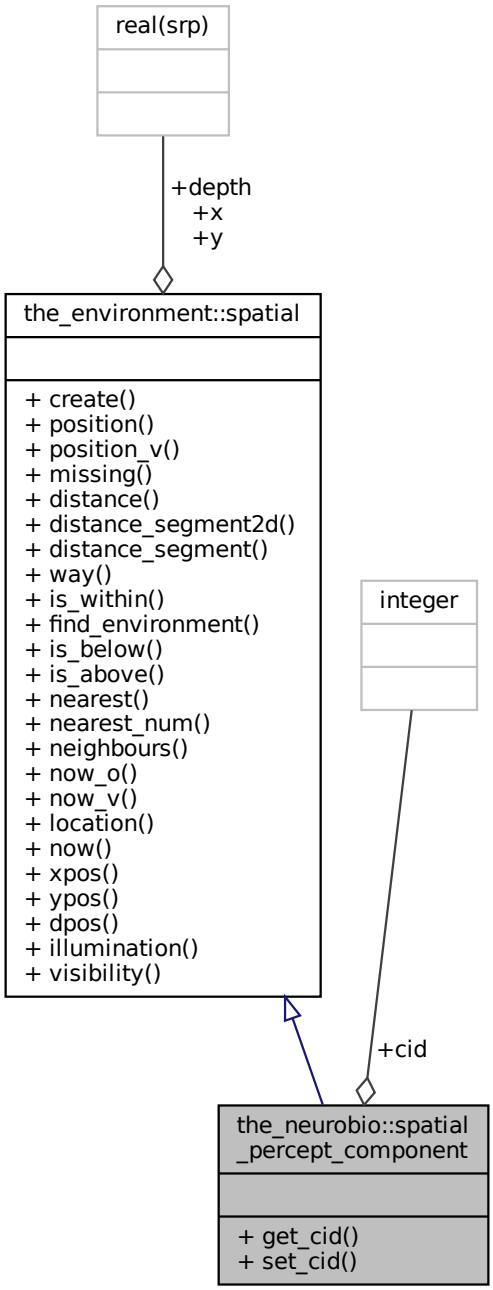
- m_env.f90

9.84 the_neurobio::spatial_percept_component Type Reference

This type defines a single spatial perception component, i.e. some single elementary spatial object that can be perceived by the agent from a big array of objects of the same type which are available in the agent's environment. Different kinds of perception objects (e.g. conspecifics, predators etc.) can be produced by extending this basic type.
Inheritance diagram for the_neurobio::spatial_percept_component:



Collaboration diagram for the_neurobio::spatial_percept_component:



Public Member Functions

- procedure, public get_cid => spatial_percept_get_cid

  *Get the unique **id** of the food item object. See `the_neurobio::spatial_percept_get_cid()`.*
- procedure, public set_cid => spatial_percept_set_cid

  *Set unique **id** for the conspecific perception component. See `the_neurobio::spatial_percept_set_cid()`.*

Public Attributes

- integer cid

  Spatial perception component adds an unique component id (`cid`) number to the basic `the_environment::spatial` so that individual objects within the whole perception object array can be identified. As a consequence, spatial perception component adds only two type-bound procedures that do the %cid.

9.84.1 Detailed Description

This type defines a single spatial perception component, i.e. some single elementary spatial object that can be perceived by the agent from a big array of objects of the same type which are available in the agent's environment. Different kinds of perception objects (e.g. conspecifics, predators etc.) can be produced by extending this basic type.

Note

Note that the **food items** the_neurobio::percept_food are implemented separately and do not currently use the spatial perception component objects. For example, individual food items are indexed seperately witghin the food resource object by `iid` data component.

Definition at line 122 of file m_neuro.f90.

9.84.2 Member Function/Subroutine Documentation

9.84.2.1 get_cid()

`procedure, public the_neurobio::spatial_percept_component::get_cid`
Get the unique **id** of the food item object. See `the_neurobio::spatial_percept_get_cid()`.
Definition at line 132 of file m_neuro.f90.

9.84.2.2 set_cid()

`procedure, public the_neurobio::spatial_percept_component::set_cid`
Set unique **id** for the conspecific perception component. See `the_neurobio::spatial_percept_set_cid()`.
Definition at line 135 of file m_neuro.f90.

9.84.3 Member Data Documentation

9.84.3.1 cid

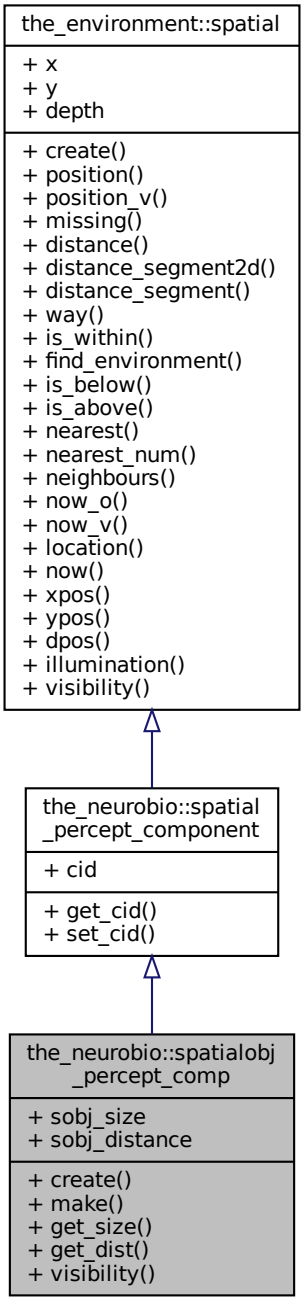
`integer the_neurobio::spatial_percept_component::cid`
Spatial perception component adds an unique component id (`cid`) number to the basic `the_environment::spatial` so that individual objects within the whole perception object array can be identified. As a consequence, spatial perception component adds only two type-bound procedures that do the %cid.
Definition at line 128 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

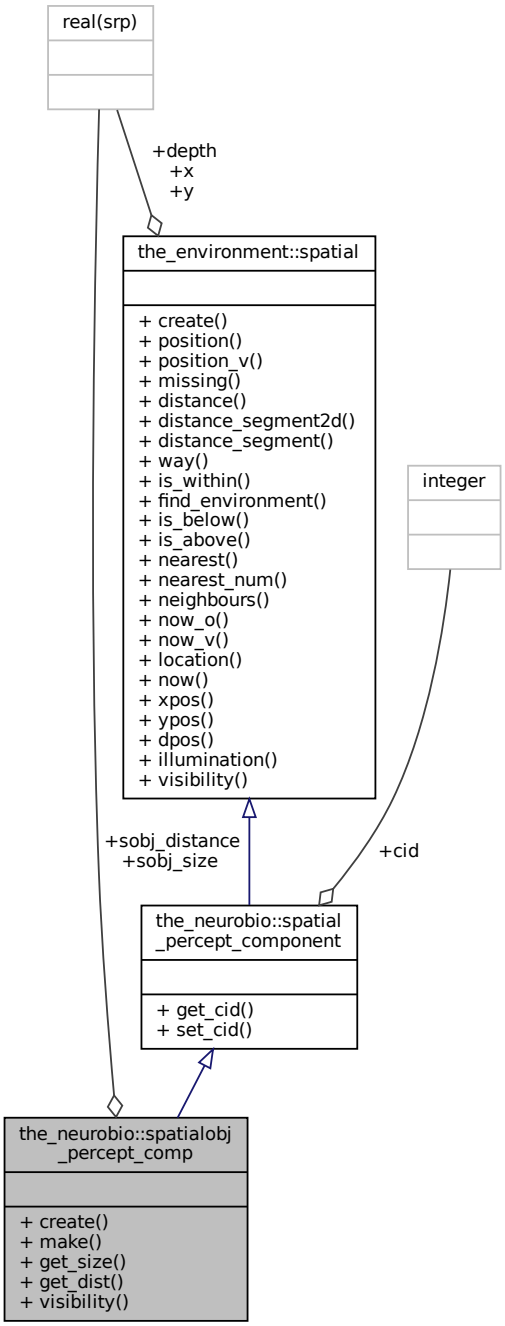
9.85 the_neurobio::spatialobj_percept_comp Type Reference

This type defines a **single** arbitrary spatial object perception component. For example, a predator perception object is then an array of such spatial object perception components.

Inheritance diagram for the_neurobio::spatialobj_percept_comp:



Collaboration diagram for the_neurobio::spatialobj_percept_comp:



Public Member Functions

- procedure, public create => spatialobj_percept_comp_create

  Create a single arbitrary spatial object perception component at an undefined position with default properties. See `the_neurobio::spatialobj_percept_comp_create()`.
- procedure, public make => spatialobj_percept_make

  *Make a single arbitrary **spatial** object perception component. See* `the_neurobio::spatialobj_percept_make()`.
- procedure, public get_size => spatialobj_percept_get_size

Get an arbitrary spatial object perception component size. See the_neurobio::spatialobj_percept_get_size().

- procedure, public get_dist => spatialobj_percept_get_dist

  Get the distance to an arbitrary spatial object perception component. See the_neurobio::spatialobj_percept_get_dist(

- procedure, public visibility => spatialobj_percept_visibility_visual_range

  Calculate the visibility range of this spatial object. Wrapper to the visual_range function. This function calculates the distance from which this object can be seen by a visual object (e.g. predator or prey). See the_neurobio::spatialobj_percept_visibility_visual_range().

Public Attributes

- real(srp) sobj_size

  Size. The size of the perception object.

- real(srp) sobj_distance

9.85.1 Detailed Description

This type defines a **single** arbitrary spatial object perception component. For example, a predator perception object is then an array of such spatial object perception components.
Definition at line 215 of file m_neuro.f90.

9.85.2 Member Function/Subroutine Documentation

9.85.2.1 create()

`procedure, public the_neurobio::spatialobj_percept_comp::create`
Create a single arbitrary spatial object perception component at an undefined position with default properties. See the_neurobio::spatialobj_percept_comp_create().
Definition at line 226 of file m_neuro.f90.

9.85.2.2 make()

`procedure, public the_neurobio::spatialobj_percept_comp::make`
Make a single arbitrary **spatial** object perception component. See the_neurobio::spatialobj_percept_make().
Definition at line 229 of file m_neuro.f90.

9.85.2.3 get_size()

`procedure, public the_neurobio::spatialobj_percept_comp::get_size`
Get an arbitrary spatial object perception component size. See the_neurobio::spatialobj_percept_get_size().
Definition at line 232 of file m_neuro.f90.

9.85.2.4 get_dist()

`procedure, public the_neurobio::spatialobj_percept_comp::get_dist`
Get the distance to an arbitrary spatial object perception component. See the_neurobio::spatialobj_percept_get_dis
Definition at line 235 of file m_neuro.f90.

9.85.2.5 visibility()

`procedure, public the_neurobio::spatialobj_percept_comp::visibility`
Calculate the visibility range of this spatial object. Wrapper to the visual_range function. This function calculates the distance from which this object can be seen by a visual object (e.g. predator or prey). See the_neurobio::spatialobj_percept_visibility_visual_range().

Definition at line 241 of file m_neuro.f90.

9.85.3 Member Data Documentation

9.85.3.1 sobj_size

`real(srp) the_neurobio::spatialobj_percept_comp::sobj_size`
Size. The size of the perception object.
Definition at line 217 of file m_neuro.f90.

9.85.3.2 sobj_distance

`real(srp) the_neurobio::spatialobj_percept_comp::sobj_distance`

Note

Any other characteristics of the perception conspecifics may be added, e.g. sex etc. What is crucial for the decision making. The distance towards this conspecific in the visual field.

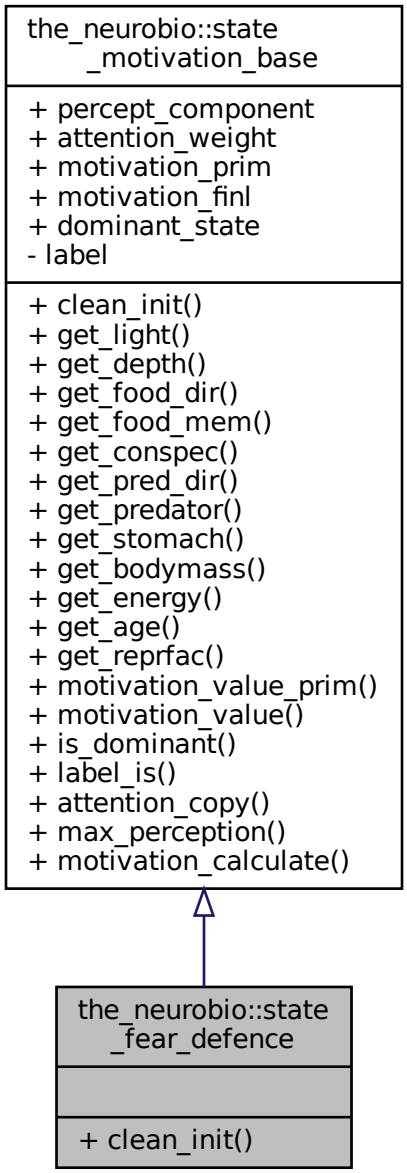
Definition at line 221 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

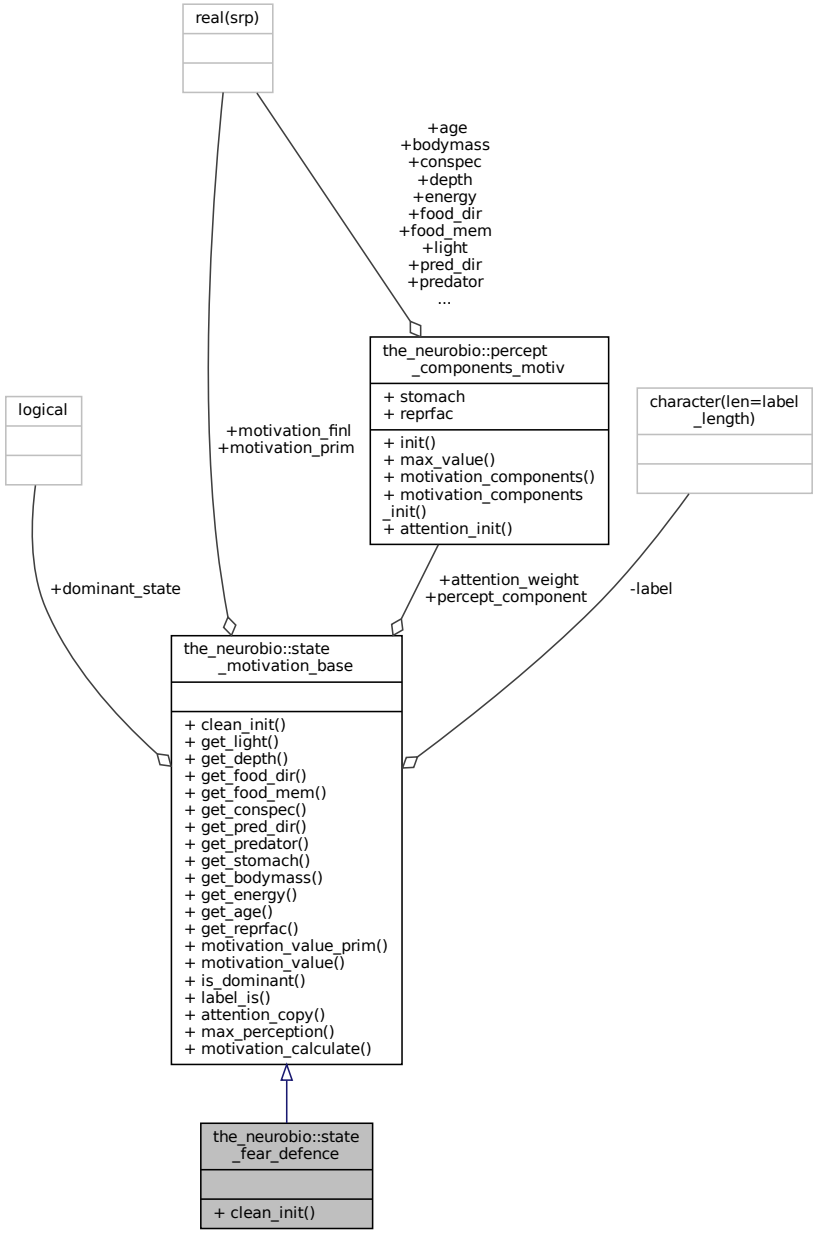
9.86 the_neurobio::state_fear_defence Type Reference

The state of **fear state**. Evokes active escape, fleeing, emigration and habitat switch.

Inheritance diagram for the_neurobio::state_fear_defence:



Collaboration diagram for the_neurobio::state_fear_defence:



Public Member Functions

- procedure, public clean_init => state_fear_defence_zero

  *Init and cleanup **fear state** motivation object. See* `the_neurobio::state_fear_defence_zero()`.

Additional Inherited Members

9.86.1 Detailed Description

The state of **fear state**. Evokes active escape, fleeing, emigration and habitat switch.
Definition at line 1078 of file m_neuro.f90.

9.86.2 Member Function/Subroutine Documentation

9.86.2.1 clean_init()

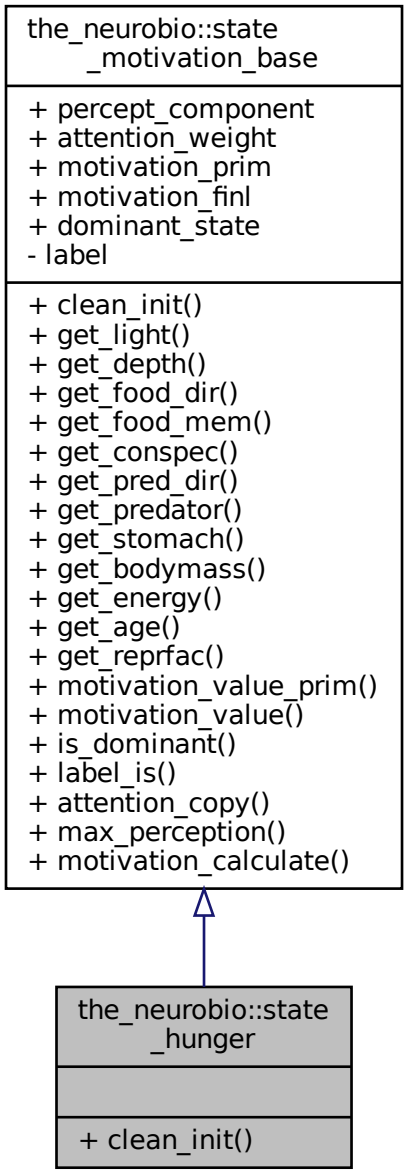
`procedure, public the_neurobio::state_fear_defence::clean_init`
Init and cleanup **fear state** motivation object. See `the_neurobio::state_fear_defence_zero()`.
Definition at line 1082 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

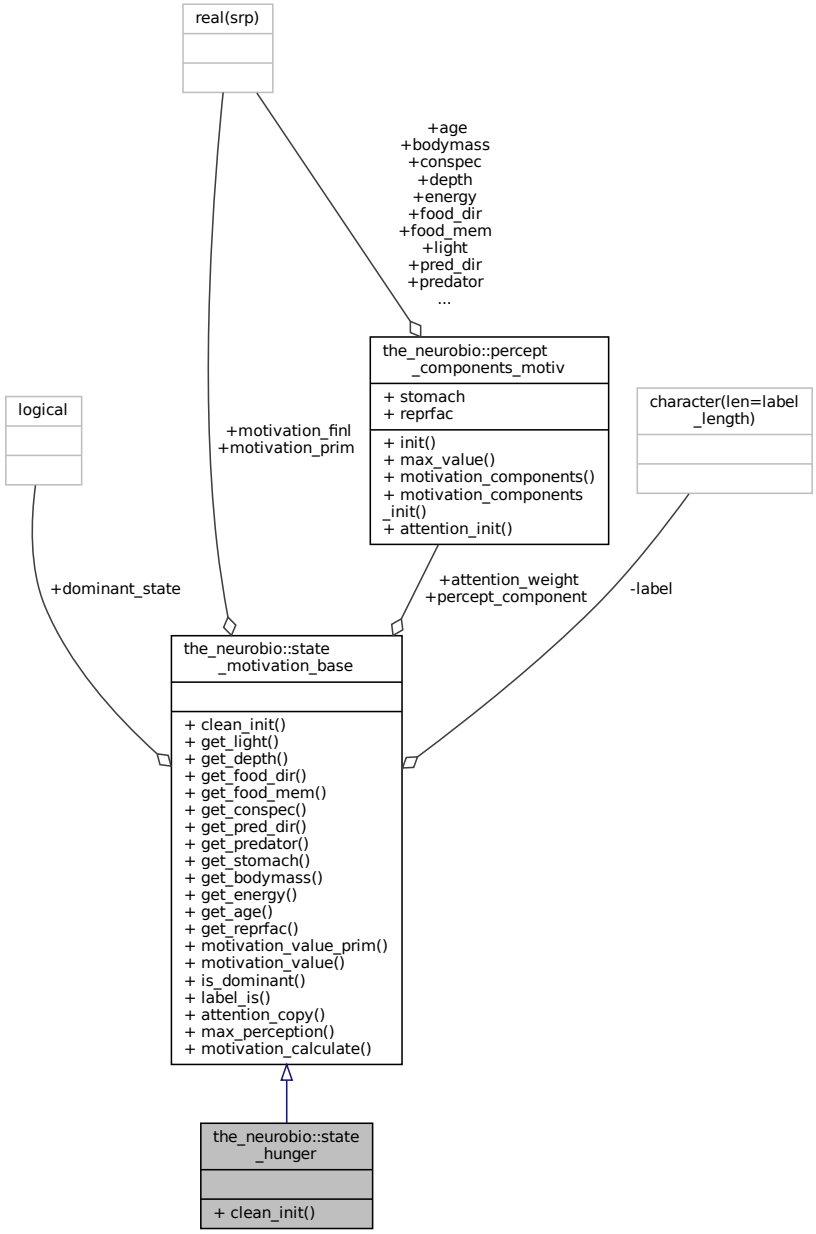
9.87 the_neurobio::state_hunger Type Reference

The motivational state of **hunger**. Evokes food seeking, eating, higher activity, emigrating and habitat switching.

Inheritance diagram for the_neurobio::state_hunger:



Collaboration diagram for the_neurobio::state_hunger:



Public Member Functions

- procedure, public clean_init => state_hunger_zero

  *Init and cleanup **hunger** motivation object. See* `the_neurobio::state_hunger_zero()`.

Additional Inherited Members

9.87.1 Detailed Description

The motivational state of **hunger**. Evokes food seeking, eating, higher activity, emigrating and habitat switching.
Definition at line 1069 of file m_neuro.f90.

9.87.2 Member Function/Subroutine Documentation

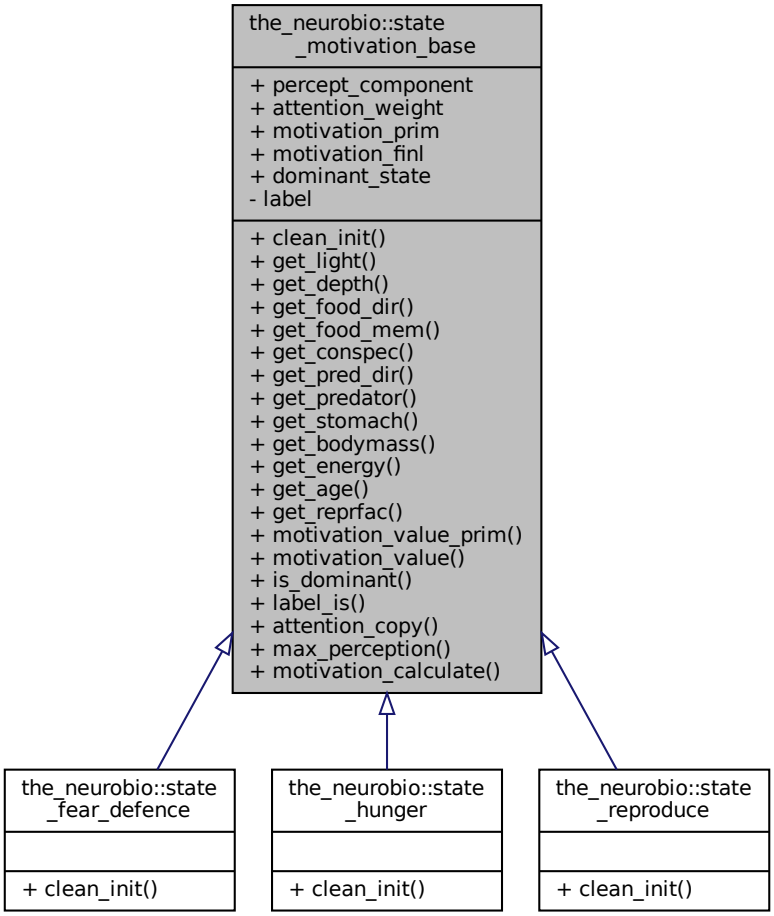
9.87.2.1 clean_init()

`procedure, public the_neurobio::state_hunger::clean_init`
Init and cleanup **hunger** motivation object. See `the_neurobio::state_hunger_zero()`.
Definition at line 1073 of file m_neuro.f90.
The documentation for this type was generated from the following file:

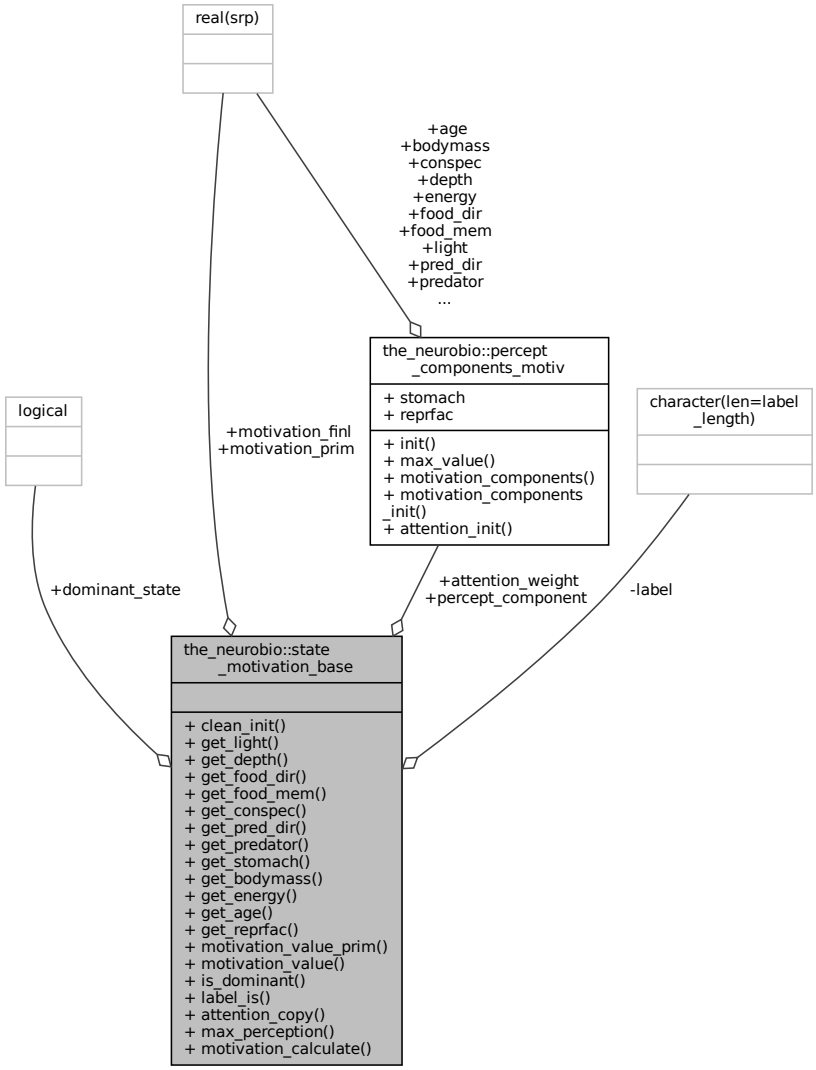
- m_neuro.f90

9.88 the_neurobio::state_motivation_base Type Reference

These types describe the **neurobiological states** of the agent. (1) Each state may have several components that are related to specific inner or outer perception objects (stimuli). (2) There is also a `motivation` component that describes the global **motivation** value for this state.
Inheritance diagram for the_neurobio::state_motivation_base:



Collaboration diagram for the_neurobio::state_motivation_base:



Public Member Functions

- procedure(motivation_init_root), deferred, public clean_init

  *Abstract **init** function that has to be overridden by each object that extends the basic motivational state type.*
- procedure, public get_light => state_motivation_light_get

  *These are basically the accessor `get`-functions, the `set`-functions are based on neural response from the perception object the_neurobio::appraisal. Get **light** perception component for this motivation state. See `the_neurobio::state_motivation_light_get()`.*
- procedure, public get_depth => state_motivation_depth_get

  *Get **depth** perception component for this motivation state. See `the_neurobio::state_motivation_depth_get()`.*
- procedure, public get_food_dir => state_motivation_food_dir_get

  *Get **directly perceived food** perception component for this motivation state. See `the_neurobio::state_motivation_food_d`*
- procedure, public get_food_mem => state_motivation_food_mem_get

  *Get **food in past memory** perception component for this motivation state. See `the_neurobio::state_motivation_food_mem`*
- procedure, public get_conspec => state_motivation_conspec_get

  *Get **conspecifics** perception component for this motivation state. See `the_neurobio::state_motivation_conspec_get()`*

- procedure, public get_pred_dir => state_motivation_pred_dir_get

  *Standard "get" function for the state neuronal **direct predation** effect component. See the_neurobio::state_motivation_pre*
- procedure, public get_predator => state_motivation_predator_get

  *Get **predator** perception component for this motivation state. See the_neurobio::state_motivation_predator_get().*
- procedure, public get_stomach => state_motivation_stomach_get

  *Get **stomach contents** perception component for this motivation state. See the_neurobio::state_motivation_stomach_ge*
- procedure, public get_bodymass => state_motivation_bodymass_get

  *Get **body mass** perception component for this motivation state. See the_neurobio::state_motivation_bodymass_get().*
- procedure, public get_energy => state_motivation_energy_get

  *Get **energy reserves** perception component for this motivation state. See the_neurobio::state_motivation_energy_get(*
- procedure, public get_age => state_motivation_age_get

  *Get **age** perception component for this motivation state. See the_neurobio::state_motivation_age_get().*
- procedure, public get_reprfac => state_motivation_reprfac_get

  *Get **reproductive factor** perception component for this motivation state. See the_neurobio::state_motivation_reprfac_g*
- procedure, public motivation_value_prim => state_motivation_motivation_prim_get

  *Get the overall **primary motivation value** (before modulation). See the_neurobio::state_motivation_motivation_prim*
- procedure, public motivation_value => state_motivation_motivation_get

  *Get the overall **final motivation value** (after modulation). See the_neurobio::state_motivation_motivation_get().*
- procedure, public is_dominant => state_motivation_is_dominant_get

  Check if the root state is the dominant state in GOS. See the_neurobio::state_motivation_is_dominant_get().
- procedure, public label_is => state_motivation_fixed_label_get

  Get the fixed label for this motivational state. Note that the label is fixed and cannot be changed. See the_neurobio::state_motivation_fixed_label_get().
- procedure, public attention_copy => state_motivation_attention_weights_transfer

  Transfer attention weights between two motivation state components. See the_neurobio::state_motivation_attention_w
- procedure, public max_perception => state_motivation_percept_maxval

  Calculate the maximum value over all perceptual components. See the_neurobio::state_motivation_percept_maxval(
- procedure, public motivation_calculate => state_motivation_calculate_prim

  *Calculate the level of the **primary motivation**. See the_neurobio::state_motivation_calculate_prim().*

Public Attributes

- type(percept_components_motiv) percept_component

  ***Perceptual components**.*
- type(percept_components_motiv) attention_weight

  ***Attention** sets the weights given to the individual perceptual components in the calculation of the motivation value.*
- real(srp) motivation_prim

  *Overall **primary motivation values**.*
- real(srp) motivation_finl

  *Overall **final** motivation value after modulation is performed.*
- logical dominant_state

  Overall GOS value, is this motivation state is dominant (TRUE/FALSE)?

Private Attributes

- character(len=label_length), private label

  *Label for the motivation state, fixed, **cannot be changed**.*

9.88.1 Detailed Description

These types describe the **neurobiological states** of the agent. (1) Each state may have several components that are related to specific inner or outer perception objects (stimuli). (2) There is also a `motivation` component that describes the global **motivation** value for this state.
This is the **base type** that serves as root for all other motivation and emotion states, which are **extensions** of this the_neurobio::state_motivation_base type.
Definition at line 952 of file m_neuro.f90.

9.88.2 Member Function/Subroutine Documentation

9.88.2.1 clean_init()

`procedure(motivation_init_root), deferred, public the_neurobio::state_motivation_base::clean↩_init`
Abstract **init** function that has to be overridden by each object that extends the basic motivational state type.

Warning

Needs abstract interface, with import of the base object type `the_neurobio::state_motivation_base`.

Definition at line 986 of file m_neuro.f90.

9.88.2.2 get_light()

`procedure, public the_neurobio::state_motivation_base::get_light`
These are basically the accessor `get`-functions, the `set`-functions are based on neural response from the perception object the_neurobio::appraisal. Get **light** perception component for this motivation state. See `the_neurobio::state_motivation_light_get()`.
Definition at line 992 of file m_neuro.f90.

9.88.2.3 get_depth()

`procedure, public the_neurobio::state_motivation_base::get_depth`
Get **depth** perception component for this motivation state. See `the_neurobio::state_motivation_depth_get()`.
Definition at line 995 of file m_neuro.f90.

9.88.2.4 get_food_dir()

`procedure, public the_neurobio::state_motivation_base::get_food_dir`
Get **directly perceived food** perception component for this motivation state. See `the_neurobio::state_motivation_food`
Definition at line 998 of file m_neuro.f90.

9.88.2.5 get_food_mem()

`procedure, public the_neurobio::state_motivation_base::get_food_mem`
Get **food in past memory** perception component for this motivation state. See `the_neurobio::state_motivation_food_`
Definition at line 1001 of file m_neuro.f90.

9.88.2.6 get_conspec()

`procedure, public the_neurobio::state_motivation_base::get_conspec`
Get **conspecifics** perception component for this motivation state. See `the_neurobio::state_motivation_conspec_get`
Definition at line 1004 of file m_neuro.f90.

9.88.2.7 get_pred_dir()

procedure, public the_neurobio::state_motivation_base::get_pred_dir
Standard "get" function for the state neuronal **direct predation** effect component. See the_neurobio::state_motivation_
Definition at line 1008 of file m_neuro.f90.

9.88.2.8 get_predator()

procedure, public the_neurobio::state_motivation_base::get_predator
Get **predator** perception component for this motivation state. See the_neurobio::state_motivation_predator_get()
Definition at line 1011 of file m_neuro.f90.

9.88.2.9 get_stomach()

procedure, public the_neurobio::state_motivation_base::get_stomach
Get **stomach contents** perception component for this motivation state. See the_neurobio::state_motivation_stomach
Definition at line 1014 of file m_neuro.f90.

9.88.2.10 get_bodymass()

procedure, public the_neurobio::state_motivation_base::get_bodymass
Get **body mass** perception component for this motivation state. See the_neurobio::state_motivation_bodymass_get
Definition at line 1017 of file m_neuro.f90.

9.88.2.11 get_energy()

procedure, public the_neurobio::state_motivation_base::get_energy
Get **energy reserves** perception component for this motivation state. See the_neurobio::state_motivation_energy_ge
Definition at line 1020 of file m_neuro.f90.

9.88.2.12 get_age()

procedure, public the_neurobio::state_motivation_base::get_age
Get **age** perception component for this motivation state. See the_neurobio::state_motivation_age_get().
Definition at line 1023 of file m_neuro.f90.

9.88.2.13 get_reprfac()

procedure, public the_neurobio::state_motivation_base::get_reprfac
Get **reproductive factor** perception component for this motivation state. See the_neurobio::state_motivation_reprfa
Definition at line 1026 of file m_neuro.f90.

9.88.2.14 motivation_value_prim()

procedure, public the_neurobio::state_motivation_base::motivation_value_prim
Get the overall **primary motivation value** (before modulation). See the_neurobio::state_motivation_motivation_pr
Definition at line 1029 of file m_neuro.f90.

9.88.2.15 motivation_value()

procedure, public the_neurobio::state_motivation_base::motivation_value
Get the overall **final motivation value** (after modulation). See the_neurobio::state_motivation_motivation_get()

Definition at line 1033 of file m_neuro.f90.

9.88.2.16 is_dominant()

`procedure, public the_neurobio::state_motivation_base::is_dominant`
Check if the root state is the dominant state in GOS. See `the_neurobio::state_motivation_is_dominant_get()`.
Definition at line 1036 of file m_neuro.f90.

9.88.2.17 label_is()

`procedure, public the_neurobio::state_motivation_base::label_is`
Get the fixed label for this motivational state. Note that the label is fixed and cannot be changed. See `the_neurobio::state_motivation_fixed_label_get()`.
Definition at line 1040 of file m_neuro.f90.

9.88.2.18 attention_copy()

`procedure, public the_neurobio::state_motivation_base::attention_copy`
Transfer attention weights between two motivation state components. See `the_neurobio::state_motivation_attentio`
Definition at line 1043 of file m_neuro.f90.

9.88.2.19 max_perception()

`procedure, public the_neurobio::state_motivation_base::max_perception`
Calculate the maximum value over all perceptual components. See `the_neurobio::state_motivation_percept_maxva`
Definition at line 1047 of file m_neuro.f90.

9.88.2.20 motivation_calculate()

`procedure, public the_neurobio::state_motivation_base::motivation_calculate`
Calculate the level of the **primary motivation**. See `the_neurobio::state_motivation_calculate_prim()`.
Definition at line 1050 of file m_neuro.f90.

9.88.3 Member Data Documentation

9.88.3.1 label

`character(len=label_length), private the_neurobio::state_motivation_base::label [private]`
Label for the motivation state, fixed, **cannot be changed**.

Note

Note that the label can be used as an **ID** for the motivational state.

Note that we cannot use `protected` attribute within derived type, so make it `private` and implement the accessor function %label_is. The label component is then set in each derived motivation object in its respective `clean_init` procedure.

Note that the `clean_init` procedure is deferred (see abstract interface) in this abstract type. Specific `clean_init` should be implemented for each of the separate motivational/emotional state type.

The procedure `motivation_components` does not seem to be necessary at this level of class hierarchy as it would duplicate that in the_neurobio::percept_components_motiv. Therefore just call the upper procedure %percept_component%motivation_components().

Definition at line 968 of file m_neuro.f90.

9.88.3.2 percept_component

`type(percept_components_motiv) the_neurobio::state_motivation_base::percept_component`
Perceptual components.
Definition at line 970 of file m_neuro.f90.

9.88.3.3 attention_weight

`type(percept_components_motiv) the_neurobio::state_motivation_base::attention_weight`
Attention sets the weights given to the individual perceptual components in the calculation of the motivation value.
Definition at line 973 of file m_neuro.f90.

9.88.3.4 motivation_prim

`real(srp) the_neurobio::state_motivation_base::motivation_prim`
Overall **primary motivation values**.
Definition at line 975 of file m_neuro.f90.

9.88.3.5 motivation_finl

`real(srp) the_neurobio::state_motivation_base::motivation_finl`
Overall **final** motivation value after modulation is performed.
Definition at line 977 of file m_neuro.f90.

9.88.3.6 dominant_state

`logical the_neurobio::state_motivation_base::dominant_state`
Overall GOS value, is this motivation state is dominant (TRUE/FALSE)?

Note

Note that only one state can be dominant at a time (Or not?

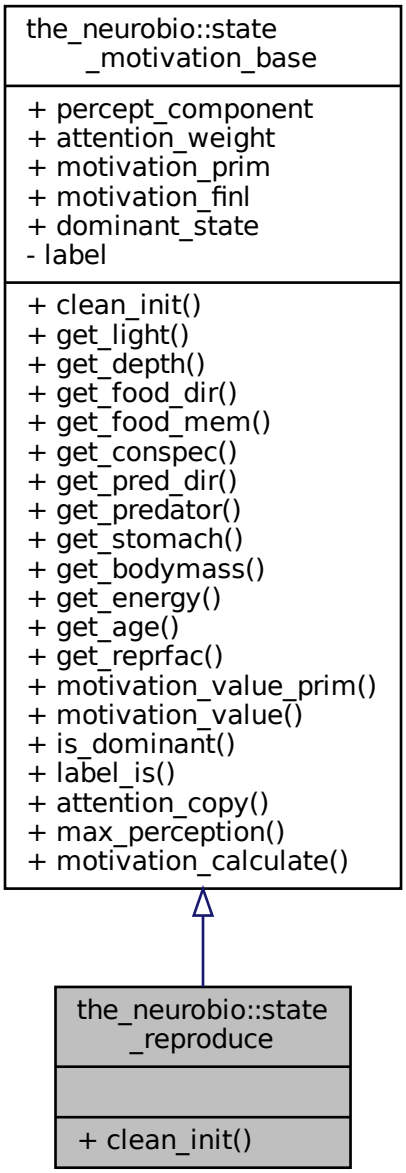
Definition at line 980 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

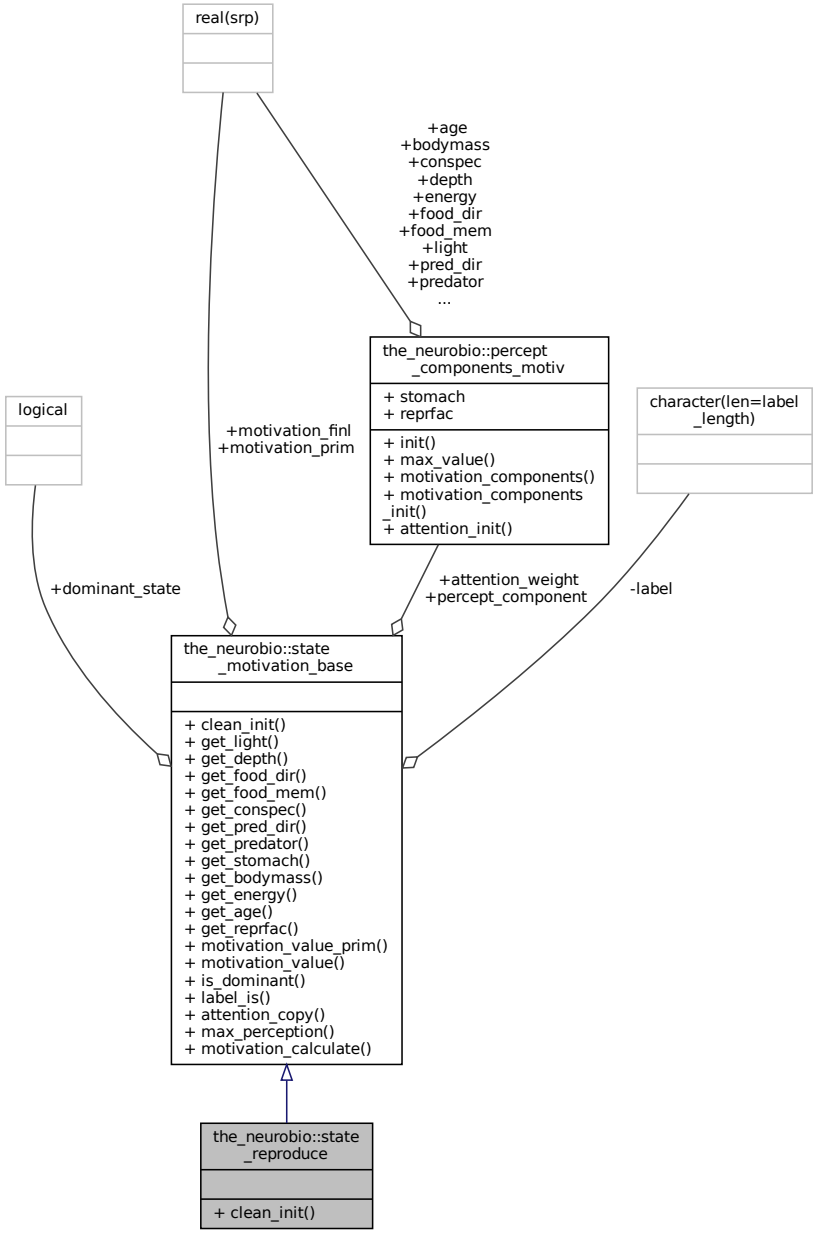
9.89 the_neurobio::state_reproduce Type Reference

The state of **reproduction**. Evokes seeking conspecifics and mating during the reproductive phase.

Inheritance diagram for the_neurobio::state_reproduce:



Collaboration diagram for the_neurobio::state_reproduce:



Public Member Functions

- procedure, public clean_init => state_reproduce_zero

  *Init and cleanup **reproductive** motivation object. See* `the_neurobio::state_reproduce_zero()`.

Additional Inherited Members

9.89.1 Detailed Description

The state of **reproduction**. Evokes seeking conspecifics and mating during the reproductive phase.
Definition at line 1087 of file m_neuro.f90.

9.89.2 Member Function/Subroutine Documentation

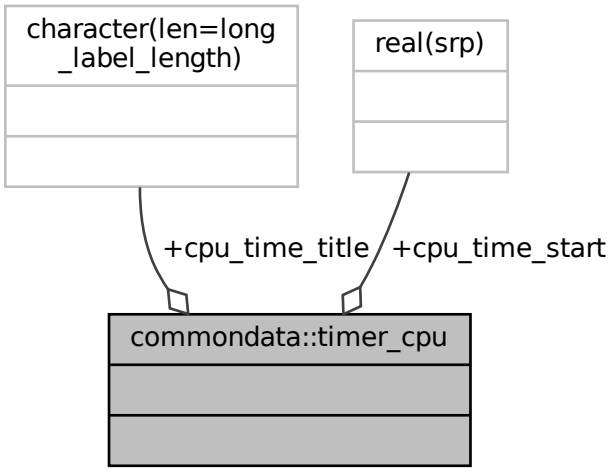
9.89.2.1 clean_init()

`procedure, public the_neurobio::state_reproduce::clean_init`
Init and cleanup **reproductive** motivation object. See `the_neurobio::state_reproduce_zero()`.
Definition at line 1091 of file m_neuro.f90.
The documentation for this type was generated from the following file:

- m_neuro.f90

9.90 commondata::timer_cpu Type Reference

CPU timer container object for debugging and speed/performance control. Arbitrary timers can be instantiated for different parts of the code and also global. Using a specific timer (`stopwatch`) is like this:
Collaboration diagram for commondata::timer_cpu:



Stopwatch Timer class

Used for execution timing of certain parts of the code.

- real(srp) cpu_time_start

  All object data components private, we should never use them directly.
- character(len=long_label_length) cpu_time_title

  Optional title for the stopwatch. Useful if we use many timers and for logger.
- procedure, public start => timer_cpu_start

  Start the timer object, stopwatch is now ON. See `commondata::timer_cpu_start()`
- procedure, public elapsed => timer_cpu_elapsed

  Calculate the time elapsed since the stopwatch subroutine was called for this instance of the timer container object. Can be called several times showing elapsed time since the grand start. See `commondata::timer_cpu_elapsed()`
- procedure, public title => timer_cpu_title

  Return the title of the current timer object. See `commondata::timer_cpu_title()`

- procedure, public show => timer_cpu_show

  A ready to use in output function that returns a formatted string for a timer combining its title and the elapsed time. See commondata::timer_cpu_show()
- procedure, public log => timer_cpu_log

  A ready to use shortcut to be used in logger, just adds the TIMER: tag in front of the normal showoutput. See commondata::timer_cpu_log()

9.90.1 Detailed Description

CPU timer container object for debugging and speed/performance control. Arbitrary timers can be instantiated for different parts of the code and also global. Using a specific timer (stopwatch) is like this:

```
call stopwatch%start("Output all agents data")
```

to **start** the stopwatch with specific title, then the function

```
... = stopwatch%elapsed()
```

returns the **elapsed time**. Then, the function stopwatch%title() outputs the title of this timer. A few other functions build on this simple functionality to provide typical shortcuts: stopwatch%show() and stopwatch%log().

Note

The near-trivial nature of this object makes it ideal for learning how to implement objects in modern Fortran.

Definition at line 1883 of file m_common.f90.

9.90.2 Member Function/Subroutine Documentation

9.90.2.1 start()

procedure, public commondata::timer_cpu::start

Start the timer object, stopwatch is now ON. See commondata::timer_cpu_start()
Definition at line 1903 of file m_common.f90.

9.90.2.2 elapsed()

procedure, public commondata::timer_cpu::elapsed

Calculate the time elapsed since the stopwatch subroutine was called for this instance of the timer container object. Can be called several times showing elapsed time since the grand start. See commondata::timer_cpu_elapsed()
Definition at line 1908 of file m_common.f90.

9.90.2.3 title()

procedure, public commondata::timer_cpu::title

Return the title of the current timer object. See commondata::timer_cpu_title()
Definition at line 1911 of file m_common.f90.

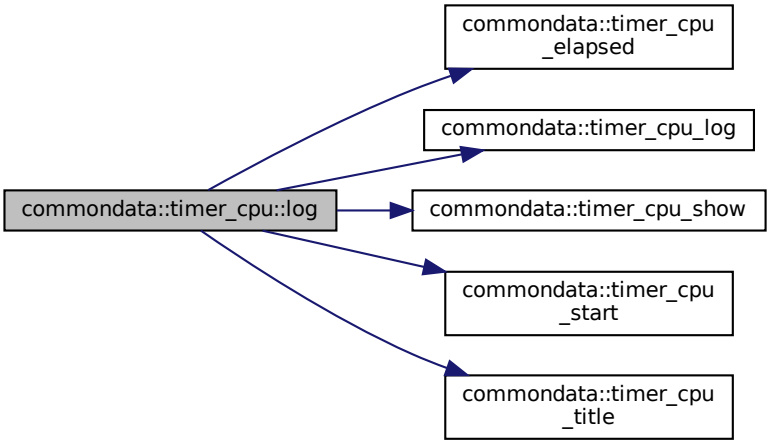
9.90.2.4 show()

procedure, public commondata::timer_cpu::show

A ready to use in output function that returns a formatted string for a timer combining its title and the elapsed time. See commondata::timer_cpu_show()
Definition at line 1915 of file m_common.f90.

9.90.2.5 log()

`procedure, public commondata::timer_cpu::log`
A ready to use shortcut to be used in logger, just adds the TIMER: tag in front of the normal `show`output. See `commondata::timer_cpu_log()`
Definition at line 1919 of file m_common.f90.
Here is the call graph for this function:



9.90.3 Member Data Documentation

9.90.3.1 cpu_time_start

`real(srp) commondata::timer_cpu::cpu_time_start`
All object data components private, we should never use them directly.
Define start time for the stopwatch.
Definition at line 1893 of file m_common.f90.

9.90.3.2 cpu_time_title

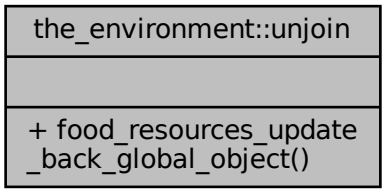
`character (len=long_label_length) commondata::timer_cpu::cpu_time_title`
Optional title for the stopwatch. Useful if we use many timers and for logger.
Definition at line 1899 of file m_common.f90.
The documentation for this type was generated from the following file:

- m_common.f90

9.91 the_environment::unjoin Interface Reference

An alias to the_environment::food_resources_update_back_global_object() method to transfer (having been modified) food resource objects out from the single united object back to the global array the_environment::global_habitats_available. See the_environment::join() for how to join an array of food resources into a single global object.

Collaboration diagram for the_environment::unjoin:



Public Member Functions

- subroutine food_resources_update_back_global_object (food_resource_collapsed, reindex)

  Transfer the (having been modified) food resource objects from the single united object `food_resource_↩collapsed` *back to the global array the_environment::global_habitats_available array. See the_environment::join() for how to join an array of food resources into a single global object.*

9.91.1 Detailed Description

An alias to the_environment::food_resources_update_back_global_object() method to transfer (having been modified) food resource objects out from the single united object back to the global array the_environment::global_habitats_available. See the_environment::join() for how to join an array of food resources into a single global object.

Warning

Note that complete restoring the food resources back to each of the individual habitat objects out of the global array must be done using the the_environment::disassemble() procedure.

Definition at line 780 of file m_env.f90.

9.91.2 Member Function/Subroutine Documentation

9.91.2.1 food_resources_update_back_global_object()

```
subroutine the_environment::unjoin::food_resources_update_back_global_object (
            type(food_resource), intent(in) food_resource_collapsed,
            logical, intent(in), optional reindex )
```

Transfer the (having been modified) food resource objects from the single united object `food_resource_↩collapsed` back to the global array the_environment::global_habitats_available array. See the_environment::join() for how to join an array of food resources into a single global object.

Parameters

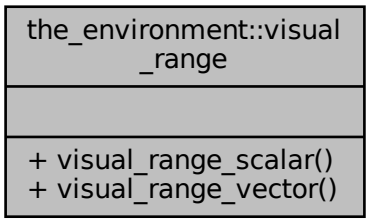
in	*food_resource_collapsed*	A collapsed food resource previously joining the input array.
in	*reindex*	reindex logical flag to reindex the unjoined resource (TRUE) upon unjoining. The default is **no** reindexing.

Definition at line 8636 of file m_env.f90.
The documentation for this interface was generated from the following file:

- m_env.f90

9.92 the_environment::visual_range Interface Reference

Calculate visual range of predator using Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`.
Collaboration diagram for the_environment::visual_range:



Public Member Functions

- real(srp) function visual_range_scalar (irradiance, prey_area, prey_contrast)

  Wrapper for calculating visual range of a fish predator using the Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`. See `srgetr()` for computational details.
- real(srp) function, dimension(size(prey_area)) visual_range_vector (irradiance, prey_area, prey_contrast_↩ vect, prey_contrast)

  Wrapper for calculating visual range of a fish predator using the Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`. See `srgetr()` for computational details.

9.92.1 Detailed Description

Calculate visual range of predator using Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`.

Note

This is a non-pure/elemental version with **debugging log output**.

Warning

The main interface name is `visual_range()`, it is this name which is used throughout the code.

Note

It is possible to use either the "debug" (this) or "fast" (next) generic interface for `visual_range()` by tweaking the interface name, e.g. to switch to the debug version rename `visual_range_debug()` to `visual_range()` and the next version to `visual_range_disable()`.

9.92.1.0.1 Specific implementations See specific implementations:

- the_environment::visual_range_scalar() for scalar argument
- the_environment::visual_range_vector() for vector argument
- the_environment::visual_range_fast() elemental (parallel-safe) version lacking sanity checks and extended debugging.

Definition at line 738 of file m_env.f90.

9.92.2 Member Function/Subroutine Documentation

9.92.2.1 visual_range_scalar()

```
real(srp) function the_environment::visual_range::visual_range_scalar (
            real(srp), intent(in) irradiance,
            real(srp), intent(in), optional prey_area,
            real(srp), intent(in), optional prey_contrast )
```

Wrapper for calculating *visual range of a fish predator* using the Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`. See `srgetr()` for computational details.

Note

Note that this is a **scalar** version. The measurement unit here is meter, might need conversion if other units are used.

Parameters

in	*irradiance*	background irradiance at specific depth
in	*prey_area*	prey area, m^2
in	*prey_contrast*	optional prey inherent contrast or default parameter if not present.

Returns

Returns visual range of the fish predator.

Example call:
```
visual_range( light_depth( 30., light_surface(100,.true.) ) )
```

9.92.2.1.1 Specific implementations

See specific implementations:

- the_environment::visual_range_scalar() for scalar argument
- the_environment::visual_range_vector() for vector argument
- the_environment::visual_range_fast() elemental (parallel-safe) version lacking sanity checks and extended debugging.

Definition at line 4730 of file m_env.f90.
Here is the call graph for this function:

9.92.2.2 visual_range_vector()

```
real(srp) function, dimension(size(prey_area)) the_environment::visual_range::visual_range_↩
vector (
            real(srp), intent(in) irradiance,
            real(srp), dimension(:), intent(in) prey_area,
```

```
real(srp), dimension(size(prey_area)), intent(in), optional prey_contrast_vect,
real(srp), intent(in), optional prey_contrast )
```

Wrapper for calculating *visual range of a fish predator* using the Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`. See `srgetr()` for computational details.

Note

This is a **vector** version, `prey_area` is mandatory and also defines the vector size for all other vector parameters including the returned function value vector. This is useful for selecting among a swarm of prey with different sizes when vector is processed. The measurement unit here is meter. Might need conversion if other units are used.

Parameters

in	*irradiance*	background irradiance at specific depth
in	*prey_area*	prey area, m^2; Mandatory parameter.
in	*prey_contrast_vect*	optional prey inherent contrast or default parameter if not present. This parameter sets **individual vector** prey contrast, so can be used for providing stochastic contrast data for each object.
in	*prey_contrast*	optional prey inherent contrast or default parameter if not present. This parameter sets **common scalar** prey contrast for the whole vector.

Returns

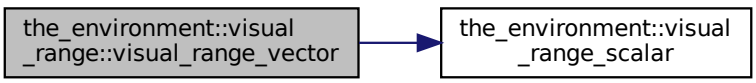
Returns visual range of the fish predator.

9.92.2.2.1 Specific implementations

See specific implementations:

- the_environment::visual_range_scalar() for scalar argument
- the_environment::visual_range_vector() for vector argument
- the_environment::visual_range_fast() elemental (parallel-safe) version lacking sanity checks and extended debugging.

Definition at line 4898 of file m_env.f90.
Here is the call graph for this function:



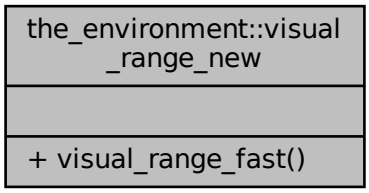
The documentation for this interface was generated from the following file:

- m_env.f90

9.93 the_environment::visual_range_new Interface Reference

Calculate visual range of predator using Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`.

Collaboration diagram for the_environment::visual_range_new:



Public Member Functions

- elemental real(srp) function visual_range_fast (irradiance, prey_area, prey_contrast)

  *Wrapper for calculating visual range of a fish predator using the Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`. This is a new **elemental** and parallel-ready visual range function wrapper making use the elemental-procedures based computational backend. See notes on `visual_range_scalar()` and `srgetr()` for computational details.*

9.93.1 Detailed Description

Calculate visual range of predator using Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`.

Note

This is a pure/elemental version with **no** debugging log output.

Warning

The main interface name is `visual_range`, it is this name which is used throughout the code.

The parameter `prey_contrast` to the **vector**-based function call must be an **scalar**. Otherwise a segmentation fault runtime error results. Vector-based call is analogous to calling `visual_range_vector()` with `prey_contrast_vect` parameter.

Definition at line 752 of file m_env.f90.

9.93.2 Member Function/Subroutine Documentation

9.93.2.1 visual_range_fast()

```
elemental real(srp) function the_environment::visual_range_new::visual_range_fast (
            real(srp), intent(in) irradiance,
            real(srp), intent(in), optional prey_area,
            real(srp), intent(in), optional prey_contrast )
```

Wrapper for calculating *visual range of a fish predator* using the Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`. This is a new **elemental** and parallel-ready visual range function wrapper making use the elemental-procedures based computational backend. See notes on `visual_range_scalar()` and `srgetr()` for computational details.

Parameters

in	*irradiance*	background irradiance at specific depth
in	*prey_area*	prey area, m^2
in	*prey_contrast*	optional prey inherent contrast or default parameter if not present.

Returns

Returns visual range of the fish predator

Warning

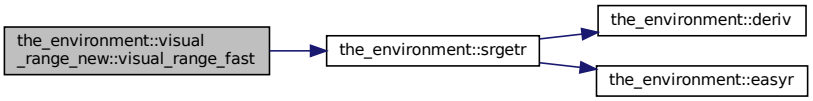
It is simplified, e.g. **no error reporting** is done. Nonetheless, debugging the old code has shown that it works okay up to the `MAX_LOG` non-whale size limit. Use the non-elemental version whenever debugging or logging is required! The parameter `prey_contrast` to the **vector**-based function call must be an **scalar**. Otherwise a segmentation fault runtime error results. Vector-based call is analogous to calling `visual_range_vector` with `prey_contrast_vect` parameter.

9.93.2.1.1 Specific implementations

See specific implementations:

- the_environment::visual_range_scalar() for scalar argument
- the_environment::visual_range_vector() for vector argument
- the_environment::visual_range_fast() elemental (parallel-safe) version lacking sanity checks and extended debugging.

Definition at line 5009 of file m_env.f90.
Here is the call graph for this function:



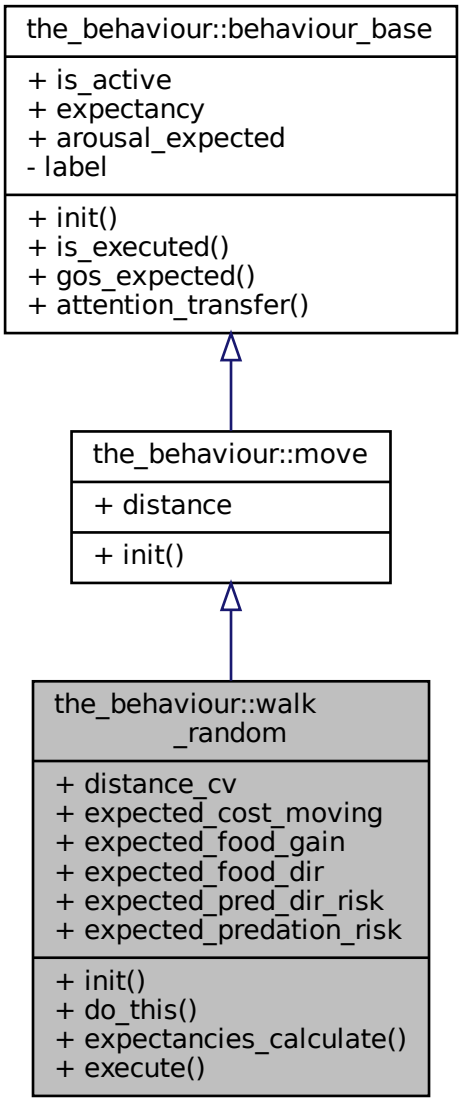
The documentation for this interface was generated from the following file:

- m_env.f90

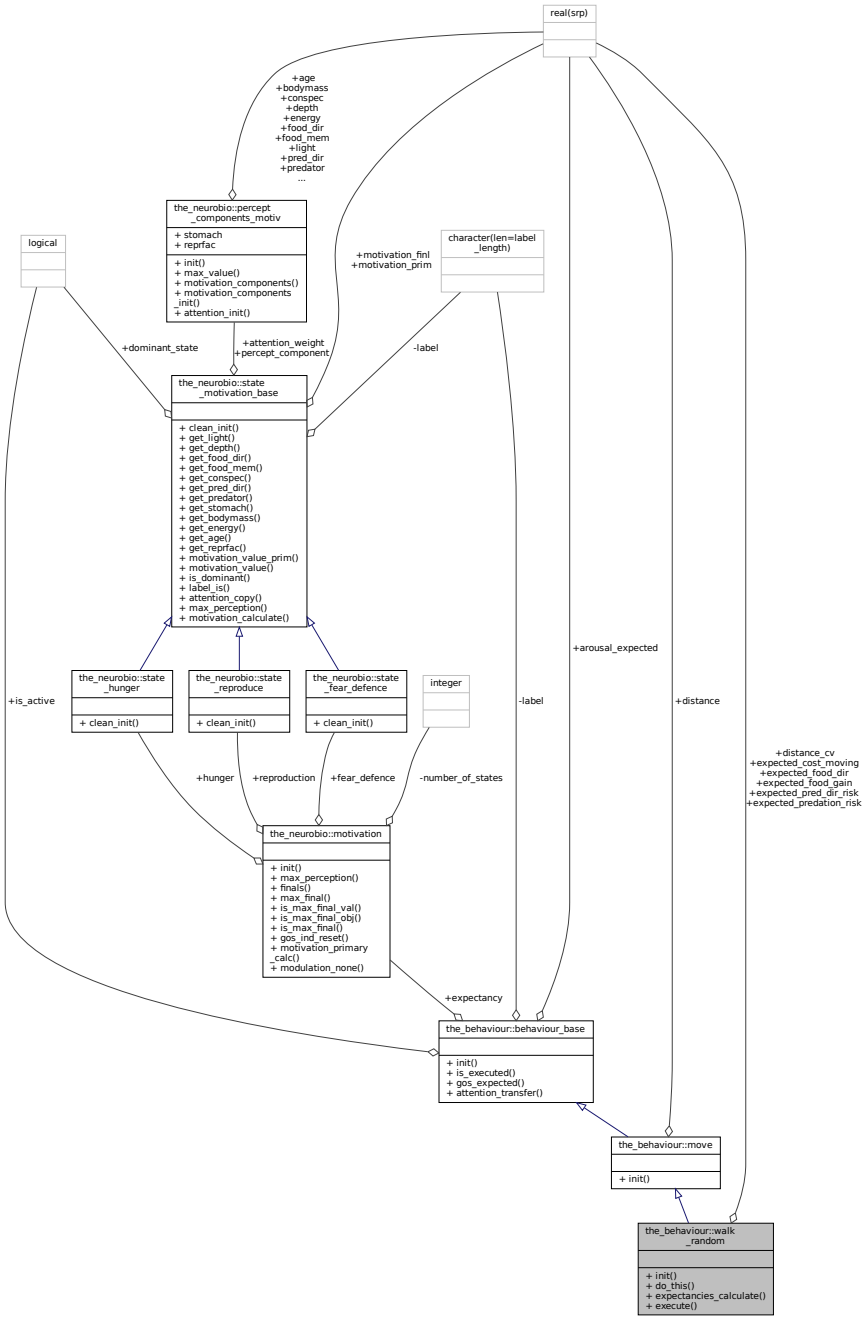
9.94 the_behaviour::walk_random Type Reference

Walk_random is a single step of a Gaussian random walk.

Inheritance diagram for the_behaviour::walk_random:



Collaboration diagram for the_behaviour::walk_random:



Public Member Functions

- procedure, public init => walk_random_init_zero

  *Initialise the **walk_random** behaviour component to a zero state. See* `the_behaviour::walk_random_init_zero()`.
- procedure, public do_this => walk_random_do_this

  The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here `the_behaviour::walk_random`*). See* `the_behaviour::walk_random_do_this()`.
- procedure, public expectancies_calculate => walk_random_motivations_expect

the_behaviour::walk_random::motivations_expect() (re)calculates motivations from fake expected perceptions following from the procedure walk_random::do_this() => the_behaviour::walk_random_do_this
See the_behaviour::walk_random_motivations_expect().

- procedure, public execute => walk_random_do_execute

  Execute this behaviour component "random walk" by this_agent agent. See the_behaviour::walk_random_do_execute(

Public Attributes

- real(srp) distance_cv

  Coefficient of variation for the Gaussian random walk.
- real(srp) expected_cost_moving

  The body mass cost of movement; depends on the distance.
- real(srp) expected_food_gain

  The expected food gain (for body mass increment) is determined from the past history for the random walk.
- real(srp) expected_food_dir

  The expected direct food perception in the novel target habitat.
- real(srp) expected_pred_dir_risk

  The expected direct predation risk is zero for random walk.
- real(srp) expected_predation_risk

  The expected general predation risk, i.e. the risk depending on the current number of predators in both the perception and memory stack.

9.94.1 Detailed Description

Walk_random is a single step of a Gaussian random walk.
Definition at line 176 of file m_behav.f90.

9.94.2 Member Function/Subroutine Documentation

9.94.2.1 init()

procedure, public the_behaviour::walk_random::init
Initialise the **walk_random** behaviour component to a zero state. See the_behaviour::walk_random_init_zero().
Definition at line 194 of file m_behav.f90.

9.94.2.2 do_this()

procedure, public the_behaviour::walk_random::do_this
The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here the_behaviour::walk_random).
See the_behaviour::walk_random_do_this().
Definition at line 201 of file m_behav.f90.

9.94.2.3 expectancies_calculate()

procedure, public the_behaviour::walk_random::expectancies_calculate
the_behaviour::walk_random::motivations_expect() (re)calculates motivations from fake expected perceptions following from the procedure walk_random::do_this() => the_behaviour::walk_random_do_t
See the_behaviour::walk_random_motivations_expect().
Definition at line 206 of file m_behav.f90.

9.94.2.4 execute()

`procedure, public the_behaviour::walk_random::execute`
Execute this behaviour component "random walk" by `this_agent` agent. See the_behaviour::walk_random_do_execu
Definition at line 210 of file m_behav.f90.

9.94.3 Member Data Documentation

9.94.3.1 distance_cv

`real(srp) the_behaviour::walk_random::distance_cv`
Coefficient of variation for the Gaussian random walk.
Definition at line 178 of file m_behav.f90.

9.94.3.2 expected_cost_moving

`real(srp) the_behaviour::walk_random::expected_cost_moving`
The body mass cost of movement; depends on the distance.
Definition at line 180 of file m_behav.f90.

9.94.3.3 expected_food_gain

`real(srp) the_behaviour::walk_random::expected_food_gain`
The expected food gain (for body mass increment) is determined from the past history for the random walk.
Definition at line 183 of file m_behav.f90.

9.94.3.4 expected_food_dir

`real(srp) the_behaviour::walk_random::expected_food_dir`
The expected direct food perception in the novel target habitat.
Definition at line 185 of file m_behav.f90.

9.94.3.5 expected_pred_dir_risk

`real(srp) the_behaviour::walk_random::expected_pred_dir_risk`
The expected direct predation risk is zero for random walk.
Definition at line 187 of file m_behav.f90.

9.94.3.6 expected_predation_risk

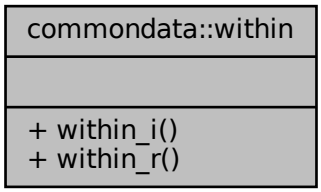
`real(srp) the_behaviour::walk_random::expected_predation_risk`
The expected general predation risk, i.e. the risk depending on the current number of predators in both the perception and memory stack.
Definition at line 190 of file m_behav.f90.
The documentation for this type was generated from the following file:

- m_behav.f90

9.95 commondata::within Interface Reference

Force a value within the range set by the vmin and vmax dummy parameter values.

Collaboration diagram for commondata::within:



Public Member Functions

- elemental integer function within_i (value_in, vmin, vmax)

  Force a value within the range set by the vmin and vmax dummy parameter values. If the value is within the range, it does not change, if it falls outside, the output force value is obtained as min(max(value, FORCE_MIN), FORCE_MAX)
- elemental real(srp) function within_r (value_in, vmin, vmax)

  Force a value within the range set by the vmin and vmax dummy parameter values. If the value is within the range, it does not change, if it falls outside, the output force value is obtained as min(max(value, FORCE_MIN), FORCE_MAX)

9.95.1 Detailed Description

Force a value within the range set by the vmin and vmax dummy parameter values.
Definition at line 5350 of file m_common.f90.

9.95.2 Member Function/Subroutine Documentation

9.95.2.1 within_i()

```
elemental integer function commondata::within::within_i (
            integer, intent(in) value_in,
            integer, intent(in), optional vmin,
            integer, intent(in) vmax )
```

Force a value within the range set by the vmin and vmax dummy parameter values. If the value is within the range, it does not change, if it falls outside, the output force value is obtained as min(max(value, FORCE_MIN), FORCE_MAX)

Parameters

in	*value_in*	Input value for forcing transformation.
in	*vmin*	minimum value of the force-to range (lower limit), if not present, a lower limit of 0.0 is used.
in	*vmax*	maximum value of the force-to range (upper limit)

Returns

an input value forced to the range.

Note

Note that this is the **integer** version of the generic `within` function.

Definition at line 5798 of file m_common.f90.

9.95.2.2 within_r()

```
elemental real(srp) function commondata::within::within_r (
            real(srp), intent(in) value_in,
            real(srp), intent(in), optional vmin,
            real(srp), intent(in) vmax )
```

Force a value within the range set by the vmin and vmax dummy parameter values. If the value is within the range, it does not change, if it falls outside, the output force value is obtained as min(max(value, FORCE_MIN), FORCE_MAX)

Parameters

in	*value_in*	Input value for forcing transformation.
in	*vmin*	minimum value of the force-to range (lower limit), if not present, a lower limit of 0.0 is used.
in	*vmax*	maximum value of the force-to range (upper limit)

Returns

an input value forced to the range.

Note

Note that this is the **real** precision version of the generic `within` function.

Definition at line 5766 of file m_common.f90.
The documentation for this interface was generated from the following file:

- m_common.f90

Chapter 10

File Documentation

10.1 m_behav.f90 File Reference

The behaviour architecture of the AHA Model.

Data Types

- type the_behaviour::behaviour_base

  Root behaviour abstract type. Several different discrete behaviours encompass the behavioural repertoire of the agent. This is the base root type from which all other behaviours are obtained by inheritance/extension.
- interface the_behaviour::behaviour_init_root

  *Abstract interface for the deferred **init** function that has to be overriden by each object that extends the basic behavioural component class.*
- type the_behaviour::move

  Movement is an umbrella abstract type linked with spatial movement.
- interface the_behaviour::move_init_root

  *Abstract interface for the deferred **init** function that has to be overriden by each object that extends the basic behavioural component class.*
- type the_behaviour::eat_food

  ***Eat food** is consuming food item(s) perceived.*
- type the_behaviour::reproduce

  Reproduce is do a single reproduction.
- type the_behaviour::walk_random

  ***Walk_random** is a single step of a Gaussian random walk.*
- type the_behaviour::freeze

  ***Freeze** is stop any locomotion completely.*
- type the_behaviour::escape_dart

  ***Escape dart** is a very fast long distance movement, normally in response to a direct predation threat.*
- type the_behaviour::approach

  ***Approach an arbitrary spatial object** is a directed movement to an arbitrary the_environment::spatial class target object.*
- type the_behaviour::approach_conspec

  ***Approach conspecifics** is directed movement towards a conspecific.*
- type the_behaviour::migrate

  ***Migrate** is move quickly directing to the other habitat*
- type the_behaviour::go_down_depth

  Go down dive deeper.
- type the_behaviour::go_up_depth

  Go up raise to a smaller depth. TODO: abstract type linking both Up and Down.
- type the_behaviour::debug_base

This is a test fake behaviour unit that is used only for debugging. It cannot be "execute"'d, but the expectancy can be calculated (normally in the debug mode).

- type the_behaviour::behaviour

  The behaviour of the agent is defined by the the_behaviour::behaviour class. This class defines the behavioural repertoire of the agent. Each of the components of the behavioural repertoire (behaviour object) is defined as a separate independent class with its own self parameter. However, the agent which performs the behaviour (the actor agent) is included as the first non-self parameter into the behaviour component methods.

- type the_behaviour::architecture_neuro

  This type is an "umbrella" for all the lower-level classes.

Modules

- module the_behaviour

  Definition of high level behavioural architecture.

Functions/Subroutines

- pure subroutine the_behaviour::behaviour_root_attention_weights_transfer (this, this_agent)

  *Transfer attention weights from the actor agent to the behaviour's GOS expectancy object. At this stage, attention weights for **this** behaviour's expectancy motivational state components are copied from the actor agent's (`this_agent`) main motivational components' attention weights.*

- elemental real(srp) function the_behaviour::behaviour_root_gos_expectation (this)

  Accessor get-function for the final expected GOS arousal from this behaviour. All calculations for are done in `expectancies_calculate` for the specific behaviour unit.

- elemental logical function the_behaviour::behaviour_root_get_is_executed (this)

  Get the execution status of the behaviour unit. If TRUE, the unit is currently active and is being executed. This is the "getter" for the_behaviour::behaviour_base::is_active.

- elemental subroutine the_behaviour::eat_food_item_init_zero (this)

  *Initialise the **eat food item** behaviour component to a zero state.*

- elemental subroutine the_behaviour::walk_random_init_zero (this)

  *Initialise the **walk_random** behaviour component to a zero state.*

- elemental subroutine the_behaviour::freeze_init_zero (this)

  *Initialise the **freeze** behaviour component to a zero state. Freeze is a special type of move to a zero distance / zero speed.*

- subroutine the_behaviour::freeze_do_this (this, this_agent)

  Do freeze by `this_agent` (the actor agent). Subjective assessment of the motivational value for this is based on the number of food items, conspecifics and predators in the perception object.

- subroutine the_behaviour::freeze_motivations_expect (this, this_agent, time_step_model, rescale_max_motivation)

  `the_behaviour::freeze::motivations_expect()` (re)calculates motivations from fake expected perceptions following from the procedure `freeze::do_this()` => `the_behaviour::freeze_do_this()`.

- subroutine the_behaviour::freeze_do_execute (this, this_agent)

  Execute this behaviour component "freeze" by `this_agent` agent.

- elemental subroutine the_behaviour::escape_dart_init_zero (this)

  *Initialise the **escape dart** behaviour component to a zero state. Dart is a quick high speed active escape.*

- subroutine the_behaviour::escape_dart_do_this (this, this_agent, predator_object, dist_is_stochastic, time_step_model)

  Do active escape dart by `this_agent` (the actor agent). Subjective assessment of the motivational value for this is based on the distance of escape (in turn, dependent on the visibility of the predator).

- subroutine the_behaviour::escape_dart_motivations_expect (this, this_agent, predator_object, time_step_model, rescale_max_motivation)

  `escape_dart::motivations_expect()` is a subroutine (re)calculating motivations from fake expected perceptions following from the procedure `escape_dart::do_this()` => `the_behaviour::escape_dart_do_this()`.

- subroutine the_behaviour::escape_dart_do_execute (this, this_agent, predator_object, environment_limits)

  Execute this behaviour component "escape" by `this_agent` agent.

- elemental subroutine the_behaviour::approach_spatial_object_init_zero (this)

  *Initialise the **approach** behaviour component to a zero state. Approach is a generic type but not abstract.*
- subroutine the_behaviour::approach_do_this (this, this_agent, target_object, target_offset, predict_window↩ _food, time_step_model)

  The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here `APPROACH`).
- subroutine the_behaviour::approach_motivations_expect (this, this_agent, target_object, target_offset, time_step_model, rescale_max_motivation)

  `the_behaviour::approach::expectancies_calculate()` (re)calculates motivations from fake expected perceptions following from the procedure `approach::do_this()` => `the_behaviour::approach_do_this()`.
- subroutine the_behaviour::approach_do_execute (this, this_agent, target_object, is_random, target_offset, environment_limits)

  Execute this behaviour component "approach" by `this_agent` agent.
- elemental subroutine the_behaviour::approach_conspecifics_init_zero (this)

  *Initialise the **approach conspecific** behaviour to a zero state. Approach conspecific is a special extension of the generic `APPROACH` behaviour.*
- subroutine the_behaviour::approach_conspecifics_do_this (this, this_agent, target_object, target_offset, predict_window_food, time_step_model)

  The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here `APPROACH_CONSPEC`).
- subroutine the_behaviour::approach_conspecifics_motivations_expect (this, this_agent, target_object, target_offset, time_step_model, rescale_max_motivation)

  `the_behaviour::approach_conspec::expectancies_calculate()` (re)calculates motivations from fake expected perceptions following from the procedure the_behaviour::approach_conspec::do_this().
- elemental subroutine the_behaviour::migrate_init_zero (this)

  *Initialise the **migrate** behaviour component to a zero state.*
- subroutine the_behaviour::migrate_do_this (this, this_agent, target_env, predict_window_food, predict_↩ window_consp, predict_window_pred, time_step_model)

  The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here `MIGRATE`).
- subroutine the_behaviour::migrate_motivations_expect (this, this_agent, target_env, predict_window_food, predict_window_consp, predict_window_pred, time_step_model, rescale_max_motivation)

  `the_behaviour::migrate::expectancies_calculate()` (re)calculates motivations from fake expected perceptions following from the procedure `migrate::do_this()`.
- subroutine the_behaviour::migrate_do_execute (this, this_agent, target_env)

  Execute this behaviour component "migrate" by `this_agent` agent.
- pure real(srp) function the_behaviour::hope (baseline, memory_old, memory_new, zero_hope, maximum_↩ hope, raw_grid_x, raw_grid_y)

  The hope function for the assessment of expectancy for a completely novel stimulus or environment for which local information is absent.
- elemental real(srp) function the_behaviour::depth_walk_default (length, walk_factor)

  Calculate the default upward and downward walk step size. This function is called from the_behaviour::go_down_do_this() and the_behaviour::go_down_motivations_expect() if the upwards or downwards walk size is not provided explicitly.
- elemental subroutine the_behaviour::go_down_depth_init_zero (this)

  *Initialise the **go down to a deeper spatial layer** behaviour component to a zero state.*
- subroutine the_behaviour::go_down_do_this (this, this_agent, max_depth, depth_walk, predict_window_↩ food, time_step_model)

  Do go down by `this_agent` (the actor agent). Subjective assessment of the motivational value for this is based on the number of food items, conspecifics and predators at the layers below the `this_agent` actor agent.
- subroutine the_behaviour::go_down_motivations_expect (this, this_agent, depth_walk, max_depth, environments, time_step_model, rescale_max_motivation)

go_down_depth::motivations_expect() is a subroutine (re)calculating motivations from fake expected perceptions following from the procedure go_down_depth::do_this() => the_behaviour::go_down_do_this().

- subroutine the_behaviour::go_down_do_execute (this, this_agent, max_depth, environments, depth_walk)

  Execute this behaviour component "go down" by this_agent agent.
- elemental subroutine the_behaviour::go_up_depth_init_zero (this)

  *Initialise the **go up to a shallower spatial layer** behaviour component to a zero state.*
- subroutine the_behaviour::go_up_do_this (this, this_agent, min_depth, depth_walk, predict_window_food, time_step_model)

  Do go up by this_agent (the actor agent). Subjective assessment of the motivational value for this is based on the number of food items, conspecifics and predators at the layers below the this_agent actor agent.
- subroutine the_behaviour::go_up_motivations_expect (this, this_agent, depth_walk, min_depth, environments, time_step_model, rescale_max_motivation)

  go_up_depth::motivations_expect() is a subroutine (re)calculating motivations from fake expected perceptions following from the procedure go_up_depth::do_this() => the_behaviour::go_up_do_this().
- subroutine the_behaviour::go_up_do_execute (this, this_agent, min_depth, environments, depth_walk)

  Execute this behaviour component "go up" by this_agent agent towards.
- elemental subroutine the_behaviour::debug_base_init_zero (this)

  *Initialise the **fake debug behaviour** behaviour component to a zero state.*
- subroutine the_behaviour::debug_base_motivations_expect (this, this_agent, time_step_model, rescale_↩ max_motivation)

  *the_behaviour::debug_base::motivations_expect() is a subroutine (re)calculating motivations from fake expected perceptions for the **fake debug behaviour**.*
- subroutine the_behaviour::eat_food_item_do_this (this, this_agent, food_item_eaten, time_step_model, distance_food_item, capture_prob, is_captured)

  *Eat a food item defined by the object food_item_eaten. The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here the_behaviour::eat_food). So, here the result of this procedure is assessment of the stomach content increment and body mass increment that would result from eating the **this** food item by the **this_agent**. The **main output** from this **do** procedure is the this behavioural unit, namely two of its internal data components:*
- subroutine the_behaviour::eat_food_item_motivations_expect (this, this_agent, food_item_eaten, time_↩ step_model, distance_food_item, capture_prob, rescale_max_motivation)

  eat_food::motivations_expect() is a subroutine (re)calculating motivations from fake expected perceptions following from the procedure eat_food::do_this() => the_behaviour::eat_food_item_do_this().
- subroutine the_behaviour::eat_food_item_do_execute (this, this_agent, food_item_eaten, food_resource_↩ real, eat_is_success)

  Execute this behaviour component "eat food item" by this_agent agent towards the food_item_eaten.
- elemental subroutine the_behaviour::reproduce_init_zero (this)

  Initialise reproduce behaviour object.
- integer function the_behaviour::maximum_n_reproductions (this)

  Calculate the maximum number of possible reproductions for this agent. It is assumed that a male can potentially fertilise several females that are within its perception object (in proximity) during a single reproduction event. For females, this number if always one.
- subroutine the_behaviour::reproduce_do_this (this, this_agent, p_reproduction, is_reproduce)

  *Do reproduce by this_agent (the actor agent) given the specific probability of successful reproduction. The probability of reproduction depends on the number of agents of the same and of the opposite sex within the visual range of the this agent weighted by the difference in the body mass between the actor agent and the average body mass of the other same-sex agents. The **main output** from this **do** procedure is the this behavioural unit object, namely its two components:*
- real(srp) function decrement_factor_fixed ()

  Calculate the decrement factor for the gonadal steroids based reproductive factor.
- subroutine the_behaviour::reproduce_motivations_expect (this, this_agent, time_step_model, reprod_prob, non_stochastic, rescale_max_motivation)

  reproduce::motivations_expect() is a subroutine (re)calculating motivations from fake expected perceptions following from reproduce::do_this() => the_behaviour::reproduce_do_this() procedure.

- subroutine the_behaviour::reproduce_do_execute (this, this_agent)

  Execute this behaviour component "reproduce" by the `this_agent` agent.
- subroutine reproduction_unsuccessful_cost_subtract ()

  Process the costs of unsuccessful reproduction. Reproduction can be unsuccessful for various reasons: insufficient reserves (reproduction results in starvation death) or stochastic no success.
- subroutine the_behaviour::walk_random_do_this (this, this_agent, distance, distance_cv, predict_window_↩ pred, predict_window_food, time_step_model)

  The "do" procedure component of the behaviour element performs the behaviour without affecting the actor agent (the_agent) and the world (here food_item_eaten) which have intent(in), so it only can change the internal representation of the behaviour (the type to which this procedure is bound to, here `WALK_RANDOM`).
- subroutine the_behaviour::walk_random_motivations_expect (this, this_agent, distance, distance_cv, predict_window_pred, predict_window_food, time_step_model, rescale_max_motivation)

  `the_behaviour::walk_random::expectancies_calculate()` (re)calculates motivations from fake expected perceptions following from the procedure `walk_random::do_this()` => `the_behaviour::walk_random_do_th
- subroutine the_behaviour::walk_random_do_execute (this, this_agent, step_dist, step_cv, environment_↩ limits)

  Execute this behaviour component "random walk" by `this_agent` agent.
- elemental subroutine, private the_behaviour::behaviour_whole_agent_init (this)

  Initialise the behaviour components of the agent, the the_behaviour::behaviour class.
- elemental subroutine the_behaviour::behaviour_whole_agent_deactivate (this)

  Deactivate all behaviour units that compose the behaviour repertoire of the agent.
- elemental character(len=label_length) function the_behaviour::behaviour_get_behaviour_label_executing (this)

  Obtain the label of the currently executing behaviour for the `this` agent.
- integer function the_behaviour::behaviour_select_conspecific (this, rescale_max_motivation)

  *Select the optimal conspecific among (possibly) several ones that are available in the **perception object** of the agent.*
- integer function the_behaviour::behaviour_select_conspecific_nearest (this)

  Select the nearest conspecific among (possibly) several ones that are available in the perception object. Note that conspecifics are sorted by distance within the perception object. Thus, this procedure just selects the first conspecific.
- integer function the_behaviour::behaviour_select_food_item (this, rescale_max_motivation)

  *Select the optimal food item among (possibly) several ones that are available in the **perception object** of the agent.*
- real(srp) function subjective_capture_prob (fitem)

  Calculate subjective probability of food item capture, as objective capture probability and random assessment error.
- integer function the_behaviour::behaviour_select_food_item_nearest (this)

  *Select the nearest food item among (possibly) several ones that are available in the perception object. This is a specific and **most simplistic** version of the `behaviour_select_food_item` function: select the nearest food item available in the agent's perception object. Because the food items are sorted within the perception object just select the first item.*
- subroutine the_behaviour::behaviour_do_eat_food_item (this, number_in_seen, food_resource_real)

  Eat a specific food item that are found in the perception object.
- subroutine the_behaviour::behaviour_do_reproduce (this)

  Reproduce based on the `this` agent's current state.
- subroutine the_behaviour::behaviour_do_walk (this, distance, distance_cv)

  Perform a random Gaussian walk to a specific average distance with certain variance (set by the CV).
- subroutine the_behaviour::behaviour_do_freeze (this)

  Perform (execute) the the_behaviour::freeze behaviour.
- subroutine the_behaviour::behaviour_do_escape_dart (this, predator_object)

  Perform (execute) the the_behaviour::escape_dart behaviour.
- subroutine the_behaviour::behaviour_do_approach (this, target_object, is_random, target_offset)

  Approach a specific the_environment::spatial class target, i.e. execute the the_behaviour::approach behaviour. The target is either a conspecific from the perception (the_neurobio::conspec_percept_comp class) or any arbitrary the_environment::spatial class object.
- subroutine the_behaviour::behaviour_do_migrate (this, target_env)

Perform (execute) the the_behaviour::migrate (migration) behaviour.

- logical function the_behaviour::behaviour_try_migrate_random (this, target_env, max_dist, prob)

  Perform a simplistic random migration. If the agent is within a specific distance to the target environment, it emigrates there with a specific fixed probability.
- subroutine the_behaviour::behaviour_do_go_down (this, depth_walk)

  Perform (execute) the the_behaviour::go_down_depth (go down) behaviour.
- subroutine the_behaviour::behaviour_do_go_up (this, depth_walk)

  Perform (execute) the the_behaviour::go_up_depth (go up) behaviour.
- elemental subroutine the_behaviour::behaviour_cleanup_history (this)

  Cleanup the behaviour history stack for the agent. All values are empty.
- subroutine the_behaviour::behaviour_select_optimal (this, rescale_max_motivation, food_resource_real)

  *Select and **execute** the optimal behaviour, i.e. the behaviour which minimizes the expected GOS arousal.*
- subroutine eat_food_select (expected_gos, selected)

  Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::eat_food behaviour unit. The subjectively optimal food item (that minimises GOS arousal) is also obtained in this procedure.
- subroutine reproduce_select (expected_gos)

  Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::reproduce behaviour unit.
- subroutine walk_random_select (expected_gos, selected)

  Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::walk_random behaviour unit. The best (subjectively optimal) walk step from the commondata::behav_walk_step_stdlen_static parameter array values (that minimises GOS arousal) is also obtained in this procedure.
- subroutine freeze_select (expected_gos)

  Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::freeze behaviour unit.
- subroutine escape_dart_select (expected_gos, selected)

  Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::escape_dart behaviour unit. The predator object that minimises the expected arousal (i.e. subjectively the most dangerous) is also obtained in this procedure.
- subroutine approach_consp_select (expected_gos, selected)

  Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::approach_conspec behaviour unit. The conspecific that minimises the expected arousal (i.e. subjectively the most attractive) is also obtained in this procedure.
- subroutine migrate_select (expected_gos, selected)

  Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::migrate behaviour unit. The habitat object that minimises the expected arousal (i.e. subjectively the most attractive) is also obtained in this procedure.
- subroutine go_down_select (expected_gos, selected)

  Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::go_down_depth behaviour unit. The vertical migration walk step, from the commondata::behav_go_up_down_step_stdlen_static parameter array, that minimises the expected arousal (i.e. subjectively optimal) is also obtained in this procedure.
- subroutine go_up_select (expected_gos, selected)

  Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::go_up_depth behaviour unit. The vertical migration walk step, from the commondata::behav_go_up_down_step_stdlen_static parameter array, that minimises the expected arousal (i.e. subjectively optimal) is also obtained in this procedure.
- subroutine debug_base_select (expected_gos)

  Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::debug_base behaviour unit.
- subroutine the_behaviour::behaviour_select_fixed_from_gos (this, rescale_max_motivation, food_↩ resource_real)

  *Select and **execute** behaviour based on the current global organismic state. This procedure is significantly different from the_behaviour::behaviour_select_optimal() in that the behaviour that is executed is not based on optimisation of the expected GOS. Rather, the current GOS fully determines which behaviour unit is executed. Such a rigid link necessarily limits the range of behaviours that could be executed.*
- elemental subroutine, private the_behaviour::neurobio_init_components (this)

  Initialise neuro-biological architecture.

Variables

- character(len= ∗), parameter, private the_behaviour::modname = "(THE_BEHAVIOUR)"

10.1.1 Detailed Description

The behaviour architecture of the AHA Model.

Author

Sergey Budaev sergey.budaev@uib.no
Jarl Giske jarl.giske@uib.no

Date

2016-2017

10.1.2 Function/Subroutine Documentation

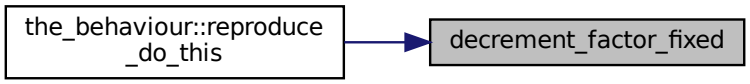
10.1.2.1 decrement_factor_fixed()

`real(srp) function reproduce_do_this::decrement_factor_fixed`
Calculate the decrement factor for the gonadal steroids based reproductive factor.

Note

This is based on fixed parameter value, trivial. A more complex pattern can also be implemented.

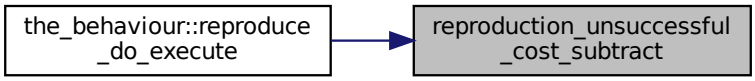
`REPRFAC_DECREMENT_FACTOR_REPRODUCTION` is a fixed decrement factor for the gonadal steroid hormone based reproductive factor (reprfact).
Definition at line 7678 of file m_behav.f90.
Here is the caller graph for this function:



10.1.2.2 reproduction_unsuccessful_cost_subtract()

`subroutine reproduce_do_execute::reproduction_unsuccessful_cost_subtract`
Process the costs of unsuccessful reproduction. Reproduction can be unsuccessful for various reasons: insufficient reserves (reproduction results in starvation death) or stochastic no success.
Unsuccessful reproduction attempt results in a cost, in terms of the body mass, that is a fraction of the normal cost of reproduction: the fraction is defined by the parameter `commondata::reproduction_cost_unsuccess` in `COMMONDATA`. The cost of unsuccessful reproduction is calculated by the function `reproduction::reproduction_cost_ur`
The body mass of the agent is then reduced to take this fraction of the full cost of reproduction. This updated value is saved into the body mass history stack (`update_history` parameter is `TRUE`).
Body length is also saved to history to make the mass and length history stack arrays synchronised.
The energy reserve of the agent, depending on both the length and the mass, is updated.
Definition at line 8233 of file m_behav.f90.

Here is the caller graph for this function:



10.1.2.3 subjective_capture_prob()

```
real(srp) function behaviour_select_food_item::subjective_capture_prob (
            integer fitem )
```

Calculate subjective probability of food item capture, as objective capture probability and random assessment error.

Note

Note that this function is contained (ower order) in the_behaviour::behaviour_select_food_item().

Then we add a random Gaussian error to the above objective value. Now we have obtained the stochastic subjective value of the capture probability for this food item including a Gaussian error. There is also a strong limitation for the subjective probability to be within the range [0.0, 1.0]. See the_neurobio::food_perception_probability_capture_memory_object() for a similar Gaussian error in subjective probability.
Definition at line 9636 of file m_behav.f90.

10.1.2.4 eat_food_select()

```
subroutine behaviour_select_optimal::eat_food_select (
            real(srp), intent(out) expected_gos,
            integer, intent(out) selected )
```

Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::eat_food behaviour unit. The subjectively optimal food item (that minimises GOS arousal) is also obtained in this procedure.

Note

This procedure is part of the_behaviour::behaviour_select_optimal() procedure and called within.

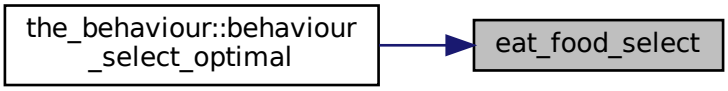
Parameters

out	*expected_gos*	expected_gos is the GOS expectancy value predicted from eating the optimal food item.
out	*selected*	selected optimal food item that would result in the minimum resulting GOS arousal.

10.1.2.4.1 Implementation details First, the the_behaviour::eat_food behaviour class is initialised by calling the the_behaviour::eat_food::init() method.
Then, perception components of the food objects are processed. If the agent has any food items in perception, then

- determine the best, optimal food item out of all the items currently in perception object of the agent: this is the food item that would result in the *minimum expected arousal* the_behaviour::behaviour::food_item_select();
- calculate the overall motivational expectancy that eating this optimal food item would provide. This value is now the *arousal expectancy* from eating behaviour (the_behaviour::eat_food) by call to the the_behaviour::eat_food::expectancies_calculate() method.

On the other hand, if the agent has no food items in its perception object, the motivational expectancy is set to a large value that is guaranteed to not win, so that this behaviour cannot be executed.
Definition at line 10746 of file m_behav.f90.
Here is the caller graph for this function:



10.1.2.5 reproduce_select()

```
subroutine behaviour_select_optimal::reproduce_select (
            real(srp), intent(out) expected_gos )
```

Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::reproduce behaviour unit.

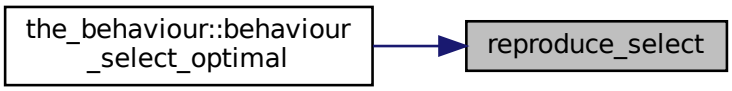
Note

This procedure is part of the_behaviour::behaviour_select_optimal() procedure and called within.

Parameters

out	*expected_gos*	expected_gos is the GOS expectancy value predicted from reproduction.

10.1.2.5.1 Implementation notes Calculation is rather straightforward here. It involves calling the the the_behaviour::reproduce::expectancies_calculate() method.
Definition at line 10800 of file m_behav.f90.
Here is the caller graph for this function:



10.1.2.6 walk_random_select()

```
subroutine behaviour_select_optimal::walk_random_select (
            real(srp), intent(out) expected_gos,
            real(srp), intent(out) selected )
```

Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::walk_random behaviour unit. The best (subjectively optimal) walk step from the commondata::behav_walk_step_stdlen_static parameter array values (that minimises GOS arousal) is also obtained in this procedure.

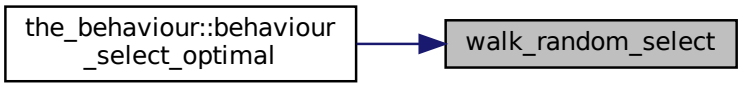
Note

This procedure is part of the_behaviour::behaviour_select_optimal() procedure and called within.

Parameters

out	*expected_gos*	expected_gos is the GOS expectancy value predicted from the Gaussian random walk of the optimal step size.
out	*selected*	selected the static step (from values in the commondata::behav_walk_step_stdlen_static array).

10.1.2.6.1 Implementation notes There are several random walks with different step sizes that are defined by the commondata::behav_walk_step_stdlen_static parameter array (i.e. a *repertoire* of walks). Therefore, selection of the arousal expectancy that would follow from the_behaviour::walk_random behaviour as a whole requires finding the *optimal walk step* among all those defined in the repertoire (commondata::behav_walk_step_stdlen_static). Such an optimal walk step size is the step size that would result in the lowest expected arousal. This is done by looping over the values of the walk step size repertoire, commondata::behav_walk_step_stdlen_static.
Definition at line 10826 of file m_behav.f90.
Here is the caller graph for this function:



10.1.2.7 freeze_select()

```
subroutine behaviour_select_optimal::freeze_select (
            real(srp), intent(out) expected_gos )
```

Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::freeze behaviour unit.

Note

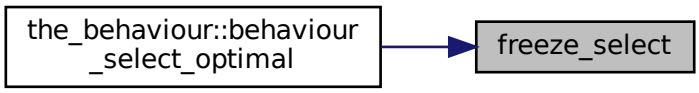
This procedure is part of the_behaviour::behaviour_select_optimal() procedure and called within.

Parameters

out	*expected_gos*	expected_gos the GOS expectancy value predicted from freezing.

10.1.2.7.1 Implementation notes First, initialise this behaviour unit object by calling the the_behaviour::freeze::init() method.
The following calculations are rather straightforward here. The arousal expectancy that would follow from freezing the_behaviour::freeze is done by calling the the_behaviour::freeze::expectancies_calculate() method.

Definition at line 10872 of file m_behav.f90.
Here is the caller graph for this function:



10.1.2.8 escape_dart_select()

```
subroutine behaviour_select_optimal::escape_dart_select (
            real(srp), intent(out) expected_gos,
            integer, intent(out) selected )
```

Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::escape_dart behaviour unit. The predator object that minimises the expected arousal (i.e. subjectively the most dangerous) is also obtained in this procedure.

Note

This procedure is part of the_behaviour::behaviour_select_optimal() procedure and called within.

Parameters

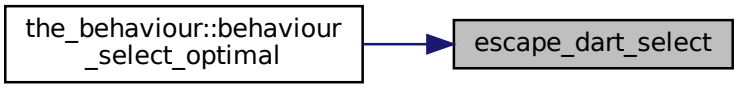
out	*expected_gos*	expected_gos is the GOS expectancy value predicted from escape movement.
out	*selected*	selected the predator object within the perception, that is associated with the lowest GOS arousal of escape, i.e. the most subjectively dangerous predator for the agent. Thus is actually the *number* of the predator within the perception object.

10.1.2.8.1 Implementation details There can be several different escape behaviour instances if the agent perceives several predators simultaneously: escape in response to each of these predators. Additionally, if the agent has no predator in the perception, escape behaviour is still possible to execute, but in such a case it is an undirected escape.
Thus, first, a check is done if the agent has any predator in perception.

- If yes, a loop is constructed overall predators within perception, the expected arousal is calculated for escape in response to each of these predators by calling the_behaviour::escape_dart::expectancies_calculate(). Finally, the predator number `selected` that minimises the expected arousal is taken as the "selected" predator and its linked (the minimum) arousal now represents the arousal expectancy for the escape behaviour.
- If there are no predators in the operception of the agent, an undirected escape is assumed. In such a case, the the_behaviour::escape_dart::expectancies_calculate() method is called omitting the optional predator object parameter. Also, the number of the predator in the perception (`selected`) is set to 0.

Definition at line 10899 of file m_behav.f90.

Here is the caller graph for this function:



10.1.2.9 approach_consp_select()

```
subroutine behaviour_select_optimal::approach_consp_select (
            real(srp), intent(out) expected_gos,
            integer, intent(out) selected )
```

Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::approach_conspec behaviour unit. The conspecific that minimises the expected arousal (i.e. subjectively the most attractive) is also obtained in this procedure.

Note

This procedure is part of the_behaviour::behaviour_select_optimal() procedure and called within.

Parameters

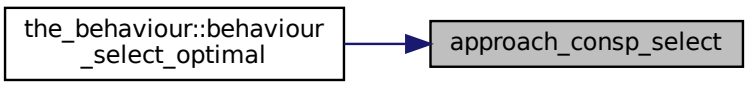
out	*expected_gos*	expected_gos is the GOS expectancy value predicted from the approach to conspecific behaviour.
out	*selected*	selected the conspecific object within the perception, that is associated with the lowest GOS arousal of approach, i.e. the most subjectively attractive conspecific for the agent. Thus is actually the *number* of the conspecific within the perception object.

10.1.2.9.1 Implementation details First, the the_behaviour::approach_conspec behaviour class is initialised by calling the the_behaviour::approach_conspec::init() method.
There can potentially be several different approach behaviour instances if the agent perceives several conspecifics simultaneously: separate instances of the approach behaviour are evaluated towards each of these conspecifics. However, if the agent has no conspecifics in its perception, approach has no mandatory target and is impossible. Thus, first, a check is done if the agent has any conspecifics in perception using the the_neurobio::perception::has_consp() method.

- If yes, determine the best, optimal conspecific to approach among all that currently are in the perception object of the agent: this is the conspecific that would result in the *minimum expected arousal* the_behaviour::behaviour::consp_select();
- calculate the overall motivational expectancy that approaching this most attractive conspecific would provide by calling the the_behaviour::approach_conspec::expectancies_calculate() method. This value is now the *arousal expectancy* from the "approach conspecifics" behaviour (the_behaviour::approach_conspec)
- On the other hand, if the agent has **no conspecifics** in its perception object, the motivational expectancy is set to a large positive value that is guaranteed to not win, so that this behaviour cannot be executed.

Definition at line 10968 of file m_behav.f90.

Here is the caller graph for this function:



10.1.2.10 migrate_select()

```
subroutine behaviour_select_optimal::migrate_select (
             real(srp), intent(out) expected_gos,
             integer, intent(out) selected )
```

Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::migrate behaviour unit. The habitat object that minimises the expected arousal (i.e. subjectively the most attractive) is also obtained in this procedure.

Note

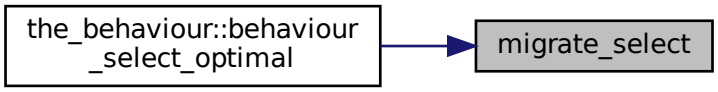
This procedure is part of the_behaviour::behaviour_select_optimal() procedure and called within.

Parameters

out	*expected_gos*	expected_gos is the GOS expectancy value predicted from migration behaviour into the optimal habitat, i.e. the habitat within the array of available habitats commondata::global_habitats_available that minimises the linked GOS arousal.
out	*selected*	selected the number of the habitat object within the commondata::global_habitats_available array, that is associated with the lowest GOS arousal of the migration behaviour, i.e. the most subjectively attractive habitat for the agent.

10.1.2.10.1 Implementation details The migration behaviour depends on the target habitat that is different than the current habitat the agent is currently in. Therefore, there can potentially be several instances of the migration behaviour with different specific migration habitat targets. Then, a loop is constructed over all these targets (they are by default obtained from the the_environment::global_habitats_available global array) and the expected arousal is calculated for each one using the_behaviour::migrate::expectancies_calculate(). Finally, the habitat that minimises the expected arousal is taken as the "selected"habitat and its linked (the minimum) arousal now represents the arousal expectancy for the migration behaviour.
Definition at line 11029 of file m_behav.f90.

Here is the caller graph for this function:



10.1.2.11 go_down_select()

```
subroutine behaviour_select_optimal::go_down_select (
            real(srp), intent(out) expected_gos,
            real(srp), intent(out) selected )
```

Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::go_down_depth behaviour unit. The vertical migration walk step, from the commondata::behav_go_up_down_step_stdlen_static parameter array, that minimises the expected arousal (i.e. subjectively optimal) is also obtained in this procedure.

Note

This procedure is part of the_behaviour::behaviour_select_optimal() procedure and called within.

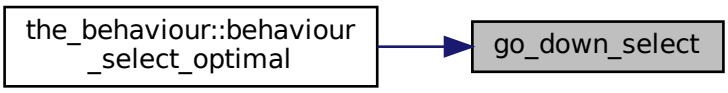
Parameters

out	*expected_gos*	expected_gos is the GOS expectancy value predicted from the downward vertical migration with the optimal step size.
out	*selected*	selected the static step size for the downwards vertical migration (from values in the commondata::behav_go_up_down_step_stdlen_static array).

10.1.2.11.1 Implementation details There are several Go down step sizes that are defined by the commondata::behav_go_up_down_step_stdlen_static parameter array (i.e. a *repertoire* of the vertical migration walks). Therefore, selection of the arousal expectancy that would follow from the_behaviour::go_down_depth behaviour as a whole requires finding the *optimal walk step* among all those defined in the repertoire (commondata::behav_go_up_down_step_stdlen_static). Such an optimal walk step size is the step size that would result in the lowest expected arousal (as computed by the_behaviour::go_down_depth::expectancies_calculate()).

- This is done by looping over the available values of the depth step size repertoire, commondata::behav_go_up_down_step_stdle

Definition at line 11093 of file m_behav.f90.
Here is the caller graph for this function:



10.1.2.12 go_up_select()

```
subroutine behaviour_select_optimal::go_up_select (
            real(srp), intent(out) expected_gos,
            real(srp), intent(out) selected )
```

Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::go_up_depth behaviour unit. The vertical migration walk step, from the commondata::behav_go_up_down_step_stdlen_static parameter array, that minimises the expected arousal (i.e. subjectively optimal) is also obtained in this procedure.

Note

This procedure is part of the_behaviour::behaviour_select_optimal() procedure and called within.

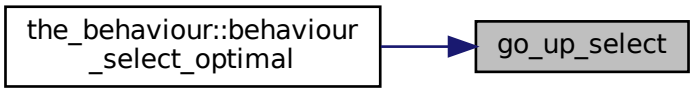
Parameters

out	*expected_gos*	expected_gos is the GOS expectancy value predicted from the upward vertical migration with the optimal step size.
out	*selected*	selected the static step size for the upwards vertical migration (from values in the commondata::behav_go_up_down_step_stdlen_static array).

10.1.2.12.1 Implementation details There are several Go up step sizes that are defined by the commondata::behav_go_up_down_step_stdlen_static parameter array (i.e. a *repertoire* of the vertical migration walks). Therefore, selection of the arousal expectancy that would follow from the_behaviour::go_up_depth behaviour as a whole requires finding the *optimal walk step* among all those defined in the repertoire (commondata::behav_go_up_down_step_stdlen_static). Such an optimal walk step size is the step size that would result in the lowest expected arousal (as computed by the_behaviour::go_up_depth::expectancies_calculate()).

- This is done by looping over the available values of the depth step size repertoire, commondata::behav_go_up_down_step_stdle

Definition at line 11150 of file m_behav.f90.
Here is the caller graph for this function:



10.1.2.13 debug_base_select()

```
subroutine behaviour_select_optimal::debug_base_select (
            real(srp), intent(out) expected_gos )
```

Calculate the expected GOS arousal that would be predicted from execution of the the_behaviour::debug_base behaviour unit.

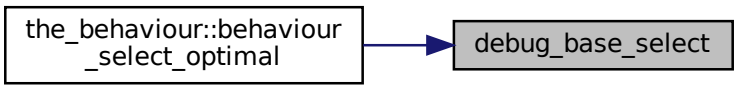
Note

This procedure is part of the_behaviour::behaviour_select_optimal() procedure and called within.

Parameters

out	*expected_gos*	expected_gos the GOS expectancy value predicted from freezing.

10.1.2.13.1 Implementation notes First, initialise this behaviour unit object by calling the the_behaviour::debug_base::init() method.
The following calculations are rather straightforward here. The arousal expectancy that would follow from the_behaviour::debug_base is done by calling the_behaviour::debug_base::expectancies_calculate().
Definition at line 11204 of file m_behav.f90.
Here is the caller graph for this function:



10.2 m_body.f90 File Reference

The Body condition and architecture of the AHA Model.

Data Types

- type the_body::condition

  CONDITION defines the physical condition of the agent
- type the_body::reproduction

  REPRODUCTION type defines parameters of the reproduction system.

Modules

- module the_body

  Definition the physical properties and condition of the agent.

Functions/Subroutines

- elemental real(srp) function the_body::length2mass (k, l)

  This is the function to calculate the body weight from the length and the Fulton condition factor (energy reserves).
- elemental real(srp) function the_body::energy_reserve (m, l)

  Calculate the current energy reserves (Fulton condition factor) from body mass and length.
- subroutine the_body::condition_init_genotype (this)

  *Initialise the individual body condition object based on the genome values. Two alleles are selected at random and input into the gamma2gene function to get the initial hormone values rescaled to 0:1. Note that the gamma2gene alleles defining the **shape** of the gamma function and the **half-max effect** are selected randomly in this version. Also, polyploid organisms are possible, in such case, two parameters are also randomly defined from a larger set (e.g. from four chromosomes in case of tetraploids). See implementation details and comments for each of the hormones.*
- subroutine the_body::birth_mortality_enforce_init_fixed_debug (this)

  This procedure enforces selective mortality of agents at birth to avoid strong selection for energy and length.
- elemental subroutine the_body::condition_clean_history (this)

  Cleanup the history stack of the body length and mass.

- elemental integer function the_body::condition_age_get (this)

  Get current age. Standard GET-function.
- elemental subroutine the_body::condition_age_reset_zero (this)

  Reset the age of the agent to zero.
- elemental subroutine the_body::condition_age_increment (this, increment)

  Increment the age of the agent by one.
- elemental real(srp) function the_body::condition_energy_current_get (this)

  Get current energy reserves. Standard GET-function.
- elemental real(srp) function the_body::condition_energy_maximum_get (this)

  Get historical maximum of energy reserves. Standard GET-function.
- elemental real(srp) function the_body::condition_body_length_get (this)

  Get current body length. Standard GET-function.
- elemental real(srp) function the_body::condition_control_unsel_get (this)

  Get current value of the control unselected trait. Standard GET-function.
- elemental real(srp) function the_body::condition_body_mass_get (this)

  Get current body mass. Standard GET-function.
- real(srp) function the_body::condition_agent_visibility_visual_range (this, object_area, contrast, time_step↩ _model)

  Calculate the visibility range of this agent. Visibility depends on the size of the agent, ambient illumination and agent contrast. Visibility is the distance from which this agent can be seen by a visual object (e.g. predator or conspecific). This function is a wrapper to the the_environment::visual_range() function.
- subroutine the_body::condition_body_mass_set_update_hist (this, value_set, update_history)

  Set body mass optionally updating the history stack.
- subroutine the_body::condition_body_length_set_update_hist (this, value_set, update_history)

  Set body length optionally updating the history stack.
- elemental real(srp) function the_body::condition_energy_birth_get (this)

  Get historical record of energy reserves at birth. Standard GET-function.
- elemental real(srp) function the_body::condition_body_length_birth_get (this)

  Get historical record of body length at birth. Standard GET-function.
- elemental real(srp) function the_body::condition_body_mass_birth_get (this)

  Get historical record of body mass at birth. Standard GET-function.
- elemental real(srp) function the_body::condition_body_mass_max_get (this)

  Get historcal maximum for body mass. Standard GET-function.
- elemental real(srp) function the_body::condition_smr_get (this)

  Get current smr. Standard GET-function.
- elemental real(srp) function the_body::condition_stomach_content_get (this)

  Get current stomach content. Standard GET-function.
- elemental real(srp) real(srp) function the_body::body_mass_processing_cost_calc_v (this, food_gain, distance_food)

  Calculate the basic processing cost of catching a food item with the mass `food_gain`.
- elemental real(srp) function the_body::condition_cost_swimming_burst (this, distance, exponent)

  The cost of swimming of a specific distance in terms of the actor's body mass.
- elemental real(srp) function the_body::body_mass_processing_cost_calc_o (this, food_obj, distance_food)

  Calculate the basic processing cost of catching a food item with the mass `food_gain`.
- elemental real(srp) function the_body::stomach_content_food_gain_fitting_v (this, food_gain, food_dist)

  Calculate the value of possible food gain as fitting into the agent's stomach, or the full gain if the food item wholly fits in.
- elemental real(srp) function the_body::stomach_content_food_gain_fitting_o (this, food_obj, food_dist)

  Calculate the value of possible food gain as fitting into the agent's stomach (or full gain if the food item fits wholly).
- elemental real(srp) function the_body::stomach_content_food_gain_non_fit_v (this, food_gain)

  Calculate extra food surplus mass non fitting into the stomach of the agent.

- elemental real(srp) function the_body::stomach_content_food_gain_non_fit_o (this, food_obj)

  Calculate extra food surplus mass non fitting into the stomach of the agent.
- elemental real(srp) function the_body::body_mass_calculate_cost_living_step (this)

  Calculate the cost of living for a single model step. So the agent mass increment per a single model step should subtract this cost.
- elemental subroutine the_body::body_mass_adjust_living_cost_step (this)

  Adjust the body mass at the end of the model step against the cost of living. We do not adjust the cost of living at each food gain as several food items can be consumed by the agent at a single time step of the model. Cost of living is now calculated at the end of the time step of the model.
- elemental subroutine the_body::body_mass_grow_do_calculate (this, food_gain, update_history)

  Do grow body mass based on food gain from a single food item adjusted for cost etc.
- elemental real(srp) function the_body::body_mass_food_processing_cost_factor_smr (this, food_gain)

  The fraction of the cost of the processing of the food item(s) depending on the agent SMR. It is scaled in terms of the ratio of the food item mass to the agent mass.
- elemental subroutine the_body::stomach_content_get_increment (this, stomach_increment)

  Do increment stomach contents with adjusted (fitted) value.
- real(srp) function the_body::body_len_grow_calculate_increment_step (this, mass_increment)

  Calculate body length increment for a time step of the model.
- subroutine the_body::body_len_grow_do_calculate_step (this, mass_increment, update_history)

  Do linear growth for one model step based on the increment function the_body::condition::len_incr().
- subroutine the_body::sex_steroids_update_increment (this)

  Update the level of the sex steroids.
- real(srp) function steroid_factor_age ()

  *Function calculating the value of the sex steroid increment factor that depends on the agent's **age**. Calculate the steroid increment factor. It is set as a nonparametric relationship that is set by a linear interpolation LINTERPOL (or DDPINTERPOL) of a parameter grid values. The increment of the sex steroid hormones depends on the **age** of the agent: it is very slight at the early stage of the ontogeny, i.e. small age, but increase to the end of the agent's lifespan.*
- real(srp) function steroid_factor_len ()

  *Function calculating the value of the sex steroid increment factor that depends on the agent's **body length**. Calculate the steroid increment factor. It is set as a nonparametric relationship that is set by a linear interpolation LINTERPOL (or DDPINTERPOL) of a parameter grid values. The increment of the sex steroid hormones depends on the **length** of the agent: it is very slight in small agents, e.g. at the early stage of the ontogeny, but increases in larger agents up to `BODY_LENGTH_MAX`.*
- elemental logical function the_body::body_mass_is_starvation_check (this)

  Check if the body mass is smaller than the birth body mass or structural body mass. An agent dies of starvation if either of these conditions is met:
- elemental logical function the_body::is_starved (body_mass, stomach_content_mass, body_mass_birth, body_mass_maximum, energy_current, energy_maximum)

  This is the backend logical function that checks if the agent is starved. It is called by the `condition::starved_death()` => `the_body::body_mass_is_starvation_check()` procedure.
- elemental subroutine the_body::stomach_content_mass_emptify_step (this)

  Digestion. Stomach contents S(t) is emptied by a constant fraction each time step.
- elemental real(srp) function the_body::stomach_emptify_backend (stomach_content_mass)

  The backend engine for calculating the stomach content mass decrement as a consequence of digestion. Stomach contents S(t) is emptied by a constant fraction each time step ΔS:
- elemental subroutine the_body::condition_energy_update_after_growth (this)

  Update the energy reserves of the agent based on its current mass and length. This subroutine should be called after any event that can change the mass or/and length of the agent, e.g. food consumption.
- elemental real(srp) function the_body::cost_swimming_standard (this, steps)

  The standard cost of swimming is a diagnostic function that shows the cost, in units of the body mass, incurred if the agent passes a distance equal to commondata::lifespan units of its body length.
- elemental real(srp) function the_body::reproduction_cost_energy_fix (this)

  *Calculate the energetic cost of reproduction in terms of the **body mass** of the this agent. The energetic cost of reproduction is obtained as a specific fixed fraction of the current body mass of the agent defined by the `commondata::reproduction_cost_body_mass` parameter.*

- real(srp) function the_body::reproduction_cost_energy_dynamic (this)

  *Calculate the energetic cost of reproduction in terms of the **body mass** of the this agent. The energetic cost of reproduction is different in males and females.*
- pure real(srp) function cost_full (offspring_mass_fact, agent_mass_fact)

  Backend function that is called in `the_body::reproduction_cost_energy_dynamic()`. Calculate the cost of reproduction as a sum of two components: (a) component that scales with the total mass of the offspring μ; (b) component that scales with the body mass of the agent M.
- pure real(srp) function cost_residual (offspring_mass_fact, agent_mass_fact)

  Backend function that is called in `the_body::reproduction_cost_energy_dynamic()`. Calculate the cost of reproduction as a sum of two components: (a) component that scales with the total mass of the offspring μ; (b) component that scales with the body mass of the agent M.
- real(srp) function the_body::reproduction_cost_unsuccessful_calc (this, cost_factor)

  Calculate the costs of unsuccessful reproduction. This is calculated as a fraction of the normal cost of reproduction returned by the function `reproduction::reproduction_cost()`. Reproduction can be unsuccessful for various reasons: insufficient reserves (reproduction results in starvation death) or stochastic no success.
- elemental subroutine the_body::reproduction_init_zero (this)

  Initialise the reproduction object for the agent. Everything is set to zero.
- elemental integer function the_body::reproduction_n_reproductions_get (this)

  Get the number of reproductions for this agent.
- elemental subroutine the_body::reproduction_n_reproductions_set (this, n_repr)

  Set the number of reproductions for the agent.
- elemental integer function the_body::reproduction_n_offspring_get (this)

  Get the number of offspring.
- elemental subroutine the_body::reproduction_n_offspring_set (this, n_offspr)

  Set the number of offspring for the agent.
- subroutine the_body::reproduction_n_increment (this, add_repr, average_mass_offspring)

  Increment the number of reproductions and offspring for this agent.
- integer function the_body::reproduction_n_offspring_calc (this, average_mass_offspr)

  Calculate the number of offspring per a single reproduction as a function of the agent's body mass.
- elemental logical function the_body::reproduction_ready_steroid_hormones_exceed (this)

  Determine if the agent's hormonal system is ready for reproduction.
- real(srp) function the_body::reproduction_mass_offspring_calc (this)

  Calculate the total mass of all offspring produced by this agent during a single reproduction event.

Variables

- character(len= $*$), parameter, private the_body::modname = "(THE_CONDITION)"

10.2.1 Detailed Description

The Body condition and architecture of the AHA Model.

Author

Sergey Budaev sergey.budaev@uib.no
Jarl Giske jarl.giske@uib.no

Date

2016-2017

10.2.2 Function/Subroutine Documentation

10.2.2.1 steroid_factor_age()

`real(srp) function sex_steroids_update_increment::steroid_factor_age`

Function calculating the value of the sex steroid increment factor that depends on the agent's **age**. Calculate the steroid increment factor. It is set as a nonparametric relationship that is set by a linear interpolation LINTERPOL (or DDPINTERPOL) of a parameter grid values. The increment of the sex steroid hormones depends on the **age** of the agent: it is very slight at the early stage of the ontogeny, i.e. small age, but increase to the end of the agent's lifespan.

Calculate the steroid increment factor. It is set as a nonparametric relationship that is set by a linear interpolation LINTERPOL (or DDPINTERPOL) of a parameter grid values. The increment of the sex steroid hormones depends on the **age** of the agent: it is very slight at the early stage of the ontogeny, i.e. small age, but increase to the end of the agent's lifespan.

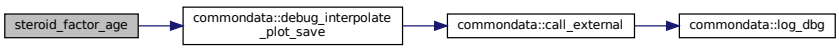
Interpolation plots can be saved in the debug mode using this plotting command: commondata::debug_interpolate_plot_save().

Warning

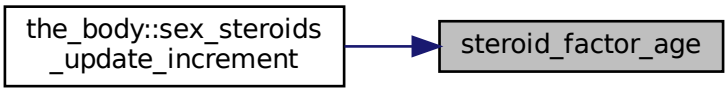
Involves **huge** number of plots, should normally be disabled.

Definition at line 1450 of file m_body.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



10.2.2.2 steroid_factor_len()

`real(srp) function sex_steroids_update_increment::steroid_factor_len`

Function calculating the value of the sex steroid increment factor that depends on the agent's **body length**. Calculate the steroid increment factor. It is set as a nonparametric relationship that is set by a linear interpolation LINTERPOL (or DDPINTERPOL) of a parameter grid values. The increment of the sex steroid hormones depends on the **length** of the agent: it is very slight in small agents, e.g. at the early stage of the ontogeny, but increases in larger agents up to `BODY_LENGTH_MAX`.

Calculate the steroid increment factor. It is set as a nonparametric relationship that is set by a linear interpolation LINTERPOL (or DDPINTERPOL) of a parameter grid values. The increment of the sex steroid hormones depends on the **length** of the agent: it is very slight in small agents, e.g. at the early stage of the ontogeny, but increases in larger agents up to `BODY_LENGTH_MAX`.

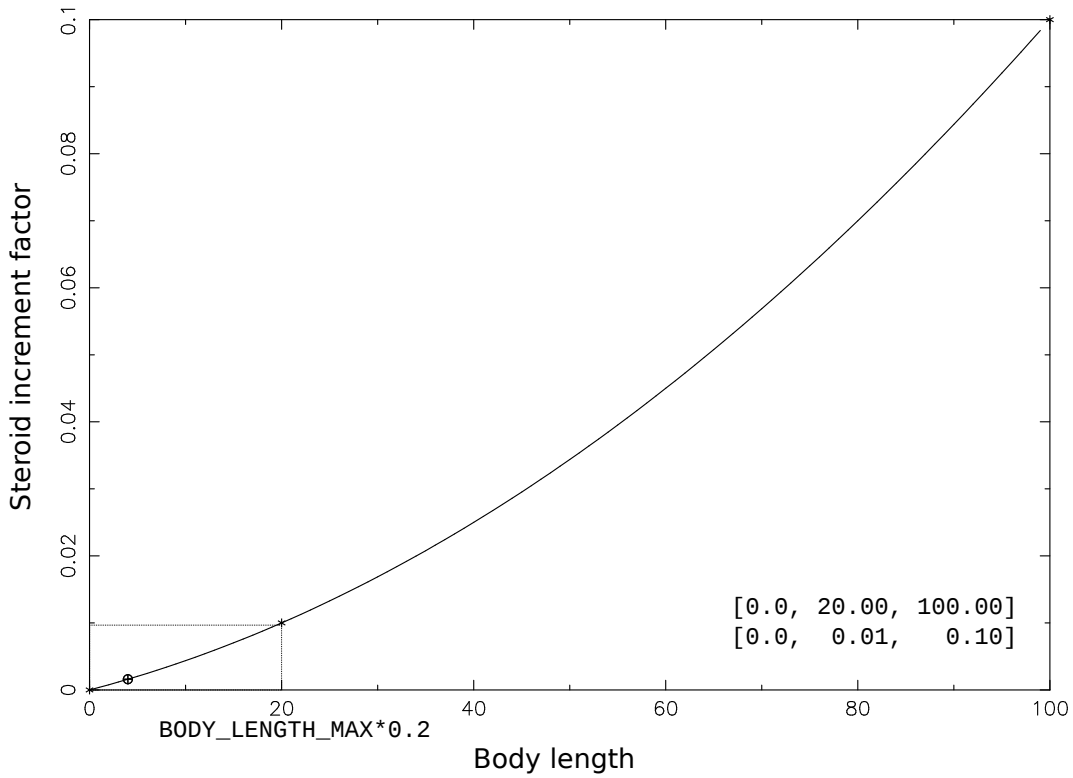


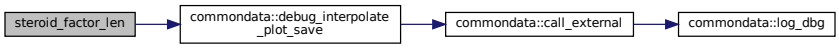
Figure 10.1 Steroid increment factor by body length

Interpolation plots can be saved in the debug mode using this plotting command: commondata::debug_interpolate_plot_save().

Warning

Involves **huge** number of plots, should normally be disabled.

Definition at line 1490 of file m_body.f90.
Here is the call graph for this function:



10.2.2.3 cost_full()

```
pure real(srp) function reproduction_cost_energy_dynamic::cost_full (
            real(srp), intent(in) offspring_mass_fact,
            real(srp), intent(in) agent_mass_fact )
```

Backend function that is called in the_body::reproduction_cost_energy_dynamic(). Calculate the cost of reproduction as a sum of two components: (a) component that scales with the total mass of the offspring μ; (b) component that scales with the body mass of the agent M.

$$C = \mu \cdot \phi + M \cdot \varphi,$$

where ϕ and φ are the scaling factors that are set by the following sex-specific parameter values: Scaling factor of the offspring mass component ϕ:

- commondata::reproduction_cost_offspring_fract_male;
- commondata::reproduction_cost_offspring_fract_female.

Scaling factor of the agent's body mass component φ:

- commondata::reproduction_cost_body_mass_factor_male;
- commondata::reproduction_cost_body_mass_factor_female.

Note

In this version, the cost component that scales with the agent's body mass is calculated from the agent's mass not subtracting the total mass of the offspring: $M \cdot \varphi$.

Parameters

in	*offspring_mass_fact*	offspring_mass_fact scaling factor of the offspring mass component ϕ: • commondata::reproduction_cost_offspring_fract_male; • commondata::reproduction_cost_offspring_fract_female.
in	*agent_mass_fact*	agent_mass_fact scaling factor of the agent's body mass component φ: • commondata::reproduction_cost_body_mass_factor_male; • commondata::reproduction_cost_body_mass_factor_female.

Returns

The cost of reproduction.

Definition at line 1801 of file m_body.f90.
Here is the caller graph for this function:

10.2.2.4 cost_residual()

```
pure real(srp) function reproduction_cost_energy_dynamic::cost_residual (
            real(srp), intent(in) offspring_mass_fact,
            real(srp), intent(in) agent_mass_fact )
```

Backend function that is called in the_body::reproduction_cost_energy_dynamic(). Calculate the cost of reproduction as a sum of two components: (a) component that scales with the total mass of the offspring μ; (b) component that scales with the body mass of the agent M.

$$C = \mu \cdot \phi + (M - \mu \cdot \phi) \cdot \varphi,$$

where ϕ and φ are the scaling factors that are set by the following sex-specific parameter values: Scaling factor of the offspring mass component ϕ:

- commondata::reproduction_cost_offspring_fract_male;
- commondata::reproduction_cost_offspring_fract_female.

Scaling factor of the agent's body mass component φ:

- commondata::reproduction_cost_body_mass_factor_male;
- commondata::reproduction_cost_body_mass_factor_female.

Note

In this version, the cost component that scales with the agent's *residual* body mass is calculated from the agent's mass after subtracting the total mass of the offspring: $(M - \mu \cdot \phi) \cdot \varphi$.

Parameters

in	*offspring_mass_fact*	offspring_mass_fact scaling factor of the offspring mass component ϕ: • commondata::reproduction_cost_offspring_fract_male; • commondata::reproduction_cost_offspring_fract_female.
in	*agent_mass_fact*	agent_mass_fact scaling factor of the agent's body mass component φ: • commondata::reproduction_cost_body_mass_factor_male; • commondata::reproduction_cost_body_mass_factor_female.

Returns

The cost of reproduction.

Definition at line 1845 of file m_body.f90.

10.3 m_common.f90 File Reference

This module defines common global parameters and objects for the AHA Model. It also contains a general overview of the AHA Model in Doxygen notation.

Data Types

- type commondata::timer_cpu

  CPU timer container object for debugging and speed/performance control. Arbitrary timers can be instantiated for different parts of the code and also global. Using a specific timer (`stopwatch`) is like this:
- interface commondata::gamma2gene

  Sigmoidal relationship between environmental factor and the organism response, as affected by the genotype and environmental error, e.g. perception and neuronal response or intrinsic baseline and phenotypic hormone levels.
- interface commondata::gene2gamma
- interface commondata::add_to_history

  Simple history stack function, add to the end of the stack. We need only to add components on top (end) of the stack and retain `HISTORY_SIZE_SPATIAL` elements of the prior history (for a spatial moving object). The stack works

as follows, assuming 100 and 200 are added:
[1 2 3 4 5 6 7 8 9 10]
*[2 3 4 5 6 7 8 9 10 **100**]*
*[3 4 5 6 7 8 9 10 100 **200**].*

- interface commondata::cm2m

  Convert cm to m.
- interface commondata::m2cm

  Convert m to cm.
- interface commondata::mm2m

  Convert mm to m.
- interface commondata::rescale

  Arbitrary rescales value(s) from one range (A:B) to another (A1:B1).
- interface commondata::within

  Force a value within the range set by the vmin and vmax dummy parameter values.
- interface commondata::is_within

  *Logical function to check if a value is within a specific range, **lower** <= **X** <= **upper**.*
- interface commondata::is_near_zero

  Checks if a real number is near 0.0. Thus function can be used for comparing two real values like this:
- interface commondata::float_equal

  Check if two real values are nearly equal using the commondata::is_near_zero(). Thus function can be used for comparing two real values like this:
- interface commondata::operator(.feq.)

  "Float equality" operator: Check if two real values are nearly equal using the commondata::is_near_zero() function. Thus function can be used for comparing two real values like the below.
- interface commondata::operator(.approx.)

  "Approximatel equality" operator: Check if two real values are approximately equal using the commondata::is_near_zero() function. Thus function can be used for comparing two real values like the below.
- interface commondata::operator(.within.)

  Interface operators `.within.` for testing whether a value (first argument) lies within the limits set by a two-element array (second argument). All the values/parameters are Fortran intrinsic types, real or integer. Usage of the operator:
- interface commondata::operator(.cat.)

  Concatenate two arrays a and b. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.
- interface commondata::average

  Calculate an average of an array excluding missing code values.
- interface commondata::is_maxval

  Check if a value is the maximum value of an array.
- interface commondata::is_minval

  Check if a value is the minimum value of an array.
- interface commondata::operator(.radd.)

  Interface operator `.radd.` performs a random addition or subtraction of two numbers with equal probability. See `commondata::random_add_subtract()`. The operator can be used as follows:

Modules

- module commondata

  COMMONDATA – definitions of global constants and procedures.

Functions/Subroutines

- elemental real(srp) function, private commondata::cm2m_r (value_cm)

  Convert cm to m.
- elemental real(hrp) function, private commondata::cm2m_hr (value_cm)

  Convert cm to m.
- elemental real(srp) function, private commondata::cm2m_i (value_cm)

  Convert cm to m.
- elemental real(srp) function, private commondata::m2cm_r (value_m)

  Convert m to cm.
- elemental real(hrp) function, private commondata::m2cm_hr (value_m)

  Convert m to cm.
- elemental real(srp) function, private commondata::m2cm_i (value_m)

  Convert m to cm.
- elemental real(srp) function, private commondata::mm2m_r (value_mm)

  Convert mm to m.
- elemental real(srp) function, private commondata::mm2m_i (value_mm)

  Convert mm to m.
- elemental real(srp) function commondata::carea (R)

  Calculate a circle area.
- elemental real(srp) function commondata::length2sidearea_fish (body_length)

  *A function linking **body length** with the body **area** in fish.*
- elemental real(srp) function, private commondata::rescale_full (value_in, A, B, A1, B1)

  Rescale a real variable with the range A:B to have the new range A1:B1.
- elemental real(srp) function, private commondata::rescale_1 (value_in, A1, B1)

  Rescale a real variable with the range 0:1 to have the new range A1:B1.
- elemental real(srp) function, private commondata::within_r (value_in, vmin, vmax)

  Force a value within the range set by the vmin and vmax dummy parameter values. If the value is within the range, it does not change, if it falls outside, the output force value is obtained as min(max(value, FORCE_MIN), FORCE_MAX)
- elemental integer function, private commondata::within_i (value_in, vmin, vmax)

  Force a value within the range set by the vmin and vmax dummy parameter values. If the value is within the range, it does not change, if it falls outside, the output force value is obtained as min(max(value, FORCE_MIN), FORCE_MAX)
- elemental logical function, private commondata::is_within_r (x, lower, upper)

  *Logical function to check if a value is within a specific range, **lower** <= **X** <= **upper**. The reverse (upper <= x <= lower) range limits can also be used; a corrective adjustment is automatically made.*
- elemental logical function, private commondata::is_within_i (x, lower, upper)

  *Logical function to check if a value is within a specific range, **lower** <= **X** <= **upper**. The reverse (upper <= x <= lower) range limits can also be used; a corrective adjustment is automatically made.*
- pure logical function, private commondata::is_within_operator_r (x, limits)

  A wrapper function for commondata::is_within() to build a user defined operator. Basically, it is the same as `is_within`, but the lower and upper limits are set as a two-element array. Usage of the operator:
- pure logical function, private commondata::is_within_operator_i (x, limits)

  A wrapper function for commondata::is_within() to build a user defined operator. Basically, it is the same as `is_within`, but the lower and upper limits are set as a two-element array. Usage of the operator:
- pure real(srp) function, private commondata::average_r (array_in, missing_code, undef_ret_null)

  Calculate an average value of a real array, excluding MISSING values.
- pure real(srp) function, private commondata::average_i (array_in, missing_code, undef_ret_null)

  Calculate an average value of an integer array, excluding MISSING values.
- real(srp) function commondata::std_dev (array_in, missing_code, undef_ret_null)

  Calculate standard deviation using trivial formula:
- pure real(srp) function, dimension(:), allocatable, private commondata::stack2arrays_r (a, b)

Concatenate two arrays a and b. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.

- pure integer function, dimension(:), allocatable, private commondata::stack2arrays_i (a, b)

  Concatenate two arrays a and b. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.
- elemental logical function, private commondata::is_near_zero_srp (test_number, epsilon)

  Checks if a real number is near 0.0. Thus function can be used for comparing two real values like the below.
- elemental logical function, private commondata::is_near_zero_hrp (test_number, epsilon)

  Checks if a real number is near 0.0. Thus function can be used for comparing two real values like the below.
- elemental logical function, private commondata::float_equal_srp (value1, value2, epsilon)

  Check if two real values are nearly equal using the commondata::is_near_zero(). Thus function can be used for comparing two real values like the below. The exact comparison (incorrect due to possible rounding):
- elemental logical function, private commondata::float_equal_hrp (value1, value2, epsilon)

  Check if two real values are nearly equal using the commondata::is_near_zero(). Thus function can be used for comparing two real values like the below. The exact comparison (incorrect due to possible rounding):
- elemental logical function, private commondata::float_equal_srp_operator (value1, value2)

  This is a wrapper for the commondata::float_equal_srp() for building the user defined operator `.feq.` with default tolerance (`epsilon` parameter). The exact real comparison (incorrect due to possible rounding):
- elemental logical function, private commondata::float_equal_hrp_operator (value1, value2)

  This is a wrapper for the commondata::float_equal_hrp() for building the user defined operator `.feq.` with default tolerance (`epsilon` parameter). The exact real comparison (incorrect due to possible rounding):
- elemental logical function, private commondata::float_approx_srp_operator (value1, value2)

  *This is a wrapper for the commondata::float_equal_srp() for building the user defined operator `.approx.` with **very high** tolerance (`epsilon` parameter). The exact real comparison (incorrect due to possible rounding):*
- elemental logical function, private commondata::float_approx_hrp_operator (value1, value2)

  *This is a wrapper for the commondata::float_equal_hrp() for building the user defined operator `.approx.` with **very high** tolerance (`epsilon` parameter). The exact real comparison (incorrect due to possible rounding):*
- subroutine commondata::do_sanitise (array, lvalid, hvalid, substval, only_wrong, tnote)

  Sanitize a real commondata::srp array, so that any value that is smaller than the minimum sensible value `lvalid` or greater then the maximum sensible value `hvalid` is substituted with `substval`. The procedure also checks the input value for IEEE validity: overflow, underflow, invalid and inexact.
- integer function commondata::ieee_error_reporting (reset, tnote)

  Check if an IEEE error condition has occurred.
- real(srp) function commondata::zeroin (ax, bx, f, tol)

  This function calculates a zero of a function f(x) in the interval (ax,bx).
- elemental real(srp) function commondata::allelescale (raw_value)

  Converts and rescales integer allele value to real value for neural response function.
- elemental real(srp) function commondata::alleleconv (raw_value)
- elemental real(srp) function commondata::cv2variance (cv, mean)

  Calculate the variance from the coefficient of variation.
- real(srp) function, private commondata::gamma2gene_additive_i4 (gs, gh, signal, erpcv)

  The function gamma2gene finds the sigmoid relationship for a complex multicomponent 2-allele impact on the neuronal response.
- real(srp) function, private commondata::gamma2gene_additive_r4 (gs, gh, signal, erpcv)

  The function gamma2gene finds the sigmoid relationship for a complex multicomponent 2-allele impact on the neuronal response.
- elemental real(srp) function, private commondata::gamma2gene_fake_vals (signal, gs, gh, n_acomps)

  This "fake" version of the `gamma2gene` is used to guess the response values in calculations.
- elemental real(srp) function, private commondata::gamma2gene_reverse (neuronal_response, gs, gh, nc)

  Reverse-calculate perception value from the given neural response value.
- pure subroutine, private commondata::add_to_history_i4 (history_array, add_this)

Simple history stack function, add to the end of the stack. We need only to add components on top of the stack and retain `commondata::history_size_spatial` elements of the prior history (for a spatial moving object). The stack works as follows, assuming 100 and 200 are added:
[1 2 3 4 5 6 7 8 9 10];
[2 3 4 5 6 7 8 9 10 100];
[3 4 5 6 7 8 9 10 100 200].

- pure subroutine, private commondata::add_to_history_r (history_array, add_this)

  Simple history stack function, add to the end of the stack. We need only to add components on top of the stack and retain `commondata::history_size_spatial` elements of the prior history (for a spatial moving object).
- pure subroutine, private commondata::add_to_history_char (history_array, add_this)

  Simple history stack function, add to the end of the stack. We need only to add components on top of the stack and retain `commondata::history_size_spatial` elements of the prior history.
- elemental integer function commondata::conv_l2i (flag, code_false, code_true)

  Converts logical to integer following a rule, default FALSE = 0, TRUE = 1.
- elemental real(srp) function commondata::conv_l2r (flag, code_false, code_true)

  Converts logical to standard (kind SRP) real, .FALSE. => 0, .TRUE. => 1.
- pure logical function, private commondata::is_maxval_r (value, array, tolerance)

  Function to check if the value is the maximum value of an array (returns TRUE), or not (return FALSE).
- pure logical function, private commondata::is_maxval_i (value, array)

  Function to check if the value is the maximum value of an array (returns TRUE), or not (return FALSE). Integer version.
- pure logical function, private commondata::is_minval_r (value, array, tolerance)

  Function to check if the value is the minimum value of an array (returns TRUE), or not (return FALSE).
- pure logical function, private commondata::is_minval_i (value, array)

  Function to check if the value is the minimum value of an array (returns TRUE), or not (return FALSE). Integer version.
- subroutine, private commondata::timer_cpu_start (this, timer_title)

  Start the timer object, stopwatch is now ON.
- real(srp) function, private commondata::timer_cpu_elapsed (this)

  Calculate the time elapsed since the stopwatch subroutine was called for this instance of the timer container object. Can be called several times showing elapsed time since the grand start.
- character(len=:) function, allocatable, private commondata::timer_cpu_title (this)

  Return the title of the current timer object.
- character(len=:) function, allocatable, private commondata::timer_cpu_show (this)

  A ready to use in output function that returns a formatted string for a timer combining its title and the elapsed time. For example: `Calculating decomposition took 20s`.
- character(len=:) function, allocatable, private commondata::timer_cpu_log (this)

  A ready to use shortcut function to be used in logger, just adds the TIMER: tag in front of the normal `show`output.
  Example use:
- subroutine commondata::call_external (command, suppress_output, suppress_error, is_background_task, cmd_is_success, exit_code)

  Call an external program using a command line. Wrapper to two alternative system shell calling intrinsic procedures.
- logical function commondata::check_external (exec)

  Check if an external procedure is executable and can be run.
- subroutine commondata::log_check_external (exec, debug_only, is_valid)

  Check if an external procedure can be called and log the result.
- subroutine commondata::debug_histogram_save (x_data, delete_csv, csv_out_file, enable_non_debug)

  *Produce a **debug plot** of histogram using an external program `hthist` from HEDTOOLS tools.*
- subroutine commondata::debug_scatterplot_save (x_data, y_data, delete_csv, csv_out_file, enable_non_↩ debug)

  *Produce a **debug plot** of 2-d scatterplot using an external program `htscatter` from HEDTOOLS tools.*
- subroutine commondata::debug_interpolate_plot_save (grid_xx, grid_yy, ipol_value, algstr, output_file, enable_non_debug)

  *Produce a **debug plot** of the **interpolation data** using an external program `htinterp` from the HEDTOOLS tools.*
- subroutine commondata::file_delete (file_name, success)

Delete a file from the local file system using Fortran open status=delete or fast POSIX C call.

- real(srp) function, private commondata::random_add_subtract (x, y)

  Random operator, adds or subtracts two values with equal probability, used in the random walk functions.
- subroutine commondata::system_init ()

  Initialises the system environment and sets basic parameters.
- subroutine commondata::system_halt (is_error, message, ignore_lockfile)

  Halt execution of the system with a specific message and exit code. The exit code is normally passed to the operating system. However, this behaviour is implementation dependent and can be unexpected on specific the platform(s) and the compiler(s).
- subroutine, private commondata::logger_init ()

  ***logger_init** Initialise the system and the system logger.*
- subroutine commondata::log_dbg (message_string, procname, modname)

  LOG_DBG: debug message to the log. The message goes to the logger only when running in the DEBUG mode.
- subroutine commondata::log_ieee (ttag, always_log, reset_flags)

  LOG_IEEE: Check and log IEEE signalling flags. Logging normally occurs only if any nonzero output from ieee_error_reporting() is found.
- character(len=:) function, allocatable, private commondata::parse_svn_version ()

  *Parse and cut revision **number** in form of string from the whole SVN revision string. SVN revision number can therefore be included into the model outputs and output file names. This is convenient because the model version is identified by a single SVN revision number.*
- character(len=long_label_length) function, dimension(:), allocatable commondata::parse_abstract (file_↩ name)

  Get and parse the model abstract. Model abstract is a short descriptive text that can span several lines and is kept in a separate file that is defined by the commondata::model_abstract_file.
- character(len=:) function, allocatable, private commondata::tag_mmdd ()

  Date (YYYYMMDD) tag for file names and logs.

Variables

Precision control for real type and IEEE float math in the model

- integer, parameter, public commondata::s_prec_32 = selected_real_kind(6, 37)

  Standard precision for real data type. We first define 32, 64 and 128 bit real kinds.
- integer, parameter, public commondata::d_prec_64 = selected_real_kind(15, 307)
- integer, parameter, public commondata::q_prec_128 = selected_real_kind(33, 4931)
- integer, parameter, public commondata::srp = S_PREC_32

  *Definition of the **standard** real type precision (SRP).*
- integer, parameter, public commondata::hrp = Q_PREC_128

  *Definition of the **high** real precision (**HRP**). This real type kind is used in pieces where a higher level of FPU precision is required, e.g. to avoid overflow/underflow and similar errors.*
- integer, parameter, public commondata::long = selected_int_kind(16)

  In some (perhaps quite rare) cases of exponentiation we may also need huge integers, those in 64 bit would probably be enough. So whenever we need such a big integer, declare it as:

Accessory parameters

- character(len= ∗), parameter, private commondata::modname = "(COMMONDATA)"

  MODNAME always refers to the name of the current module for use by the LOGGER function LOG_DBG. Note that in the debug mode (if IS_DEBUG=TRUE) LOGGER should normally produce additional messages that are helpful for debugging and locating possible sources of errors. MODNAME is declared private and is not accessible outside of this module. Each procedure should also have a similar private constant `commondata::procname`.
- character(len= ∗), parameter, private commondata::procname = ""

  PROCNAME is the procedure name for logging and debugging (with `commondata::modname`).
- character(len= ∗), parameter, public commondata::svn_version_string = "$Revision: 9552 $"

  ***Subversion** or Mercurial revision number (or ID) of the model code.*
- character(len=:), allocatable, public, protected commondata::svn_version

Subversion *or Mercurial revision number that is parsed by* `commondata::parse_svn_version()`*. It is shorter than* `commondata::svn_version_string` *and does not contain blanks. Therefore, it can be used for building output file names.*

- logical, parameter, public commondata::true =.TRUE.

  Safety parameter avoid errors in logical values, so we can now refer to standard Fortran `.TRUE.` *and* `.FALSE.` *as* `YES` *and* `NO` *or* `TRUE` *and* `FALSE`
- logical, parameter, public commondata::false =.FALSE.
- logical, parameter, public commondata::yes =.TRUE.
- logical, parameter, public commondata::no =.FALSE.
- real(srp), parameter, public commondata::zero = epsilon(0.0_SRP)

  Some parameters should never be zero or below. In such cases they could be set to some smallest distinguishable non-zero value. Here set as the Fortran intrinsic `epsilon` *function, a value that is almost negligible compared to one, i.e. the smallest* `real` *number E such that* $1 + E > 1$*. In some cases it is also reasonable to set the tolerance limit to this parameter (see Float point computations).*
- real(srp), parameter, public commondata::tiny_srp =tiny(1.0_SRP)

  The smallest positive number in the commondata::srp standard real model.
- real(hrp), parameter, public commondata::tiny_hrp =tiny(1.0_HRP)

  The smallest positive number in the commondata::hrp high precision real model. See Float point computations.
- real(srp), parameter, public commondata::lo_valid_sanitised = TINY_SRP $*$ 10.0_SRP

  Lower bound for do_sanitise() procedure. This is the lowest value that considered valid.
- real(srp), parameter, public commondata::hi_valid_sanitised = huge(1.0_SRP)/100.0_SRP

  Higher bound for do_sanitise() procedure. This is the highest value that considered valid.
- real(srp), parameter, public commondata::tolerance_low_def_srp = TINY_SRP $*$ 5.0_SRP

  Default value of low tolerance (high precision). This is the standard commondata::srp precision. See Float point computations.
- real(hrp), parameter, public commondata::tolerance_low_def_hrp = TINY_HRP $*$ 5.0_HRP

  Default value of low tolerance (high precision). This is the high commondata::hrp precision. See Float point computations.
- real(srp), parameter, public commondata::tolerance_high_def_srp = ZERO $*$ 1000.0_SRP

  Default value of high tolerance (low precision). This is the standard commondata::srp precision real. See Float point computations.
- real(hrp), parameter, public commondata::tolerance_high_def_hrp = epsilon(0.0_HRP) $*$ 1000.0_HRP

  Default value of high tolerance (low precision). This is the high commondata::hrp precision real. See Float point computations.
- real(srp), parameter, public commondata::missing = -9999.0_SRP

  Numerical code for missing and invalid ***real type*** *values.*
- real(srp), parameter, public commondata::invalid = -9999.0_SRP
- integer, parameter, public commondata::unknown = -9999

  Numerical code for invalid or missing ***integer*** *counts.*
- real(srp), parameter, public commondata::pi =4.0_SRP$*$atan(1.0_SRP)

  The ***PI*** *number.*
- character(len= $*$), parameter, public commondata::csv =".csv"

  Standard data file extension for data output is now .csv.
- character(len= $*$), parameter, public commondata::ps =".ps"

  Standard file extension for debug and other PostScript plots.
- integer, parameter, public commondata::filename_length = 255

  Set the standard length of the file name, are 255 characters enough?
- logical, parameter, public commondata::use_posix_fs_utils = .TRUE.

  Logical flag for setting if POSIX direct filesystem procedures are used. These utilities are implemented in HED-TOOLS for standard POSIX C call via the Fortran interface. They should work safer, better and faster than indirect procedure wrappers (e.g. calling `system()`*) but are not fully portable and might not work as expected on all systems and compilers.*
- integer, parameter, public commondata::label_length = 14

  The length of standard character string labels. We use labels for various objects, like alleles, perceptual and neural components / bundles etc. For simplicity, they all have the same length. It should be big enough to fit the longest whole label.
- integer, parameter, public commondata::long_label_length = 128

  The length of long labels.
- integer, parameter, public commondata::label_cst = 97

  This parameter defines the range of characters that is used for generating random labels, 97:122 corresponds to lowercase Latin letters.

- integer, parameter, public commondata::label_cen = 122
- character(len= *), parameter commondata::lock_file = "lock_simulation_running.lock"

  The name of the lock file. The lock file is created at the start of the simulation and is deleted at the end of the simulation. It can be used to signal that simulation is still ongoing to external utilities and scripts. See The lock file.
- integer, public, protected commondata::lock_file_unit

  This is the unit number that identifies the lock file. The lock file is created at the start of the simulation and is deleted at the end of the simulation. It can be used to signal that simulation is still ongoing to external utilities and scripts. See The lock file.
- character(len= *), parameter commondata::stop_file = "stop_simulation_running.lock"

  The name of the stop file. The stop file is checked before each new generation of the Genetic Algorithm. If this file is found, simulation does not go to the next generation and just stops. See The stop file.
- integer, parameter, public commondata::platform_windows = 100

  Runtime platform ID constants. Use these constants for determining the current runtime platform, e.g. `Platform_Running = PLATFORM_WINDOWS`. See `commondata::platform_running`.
- integer, parameter, public commondata::platform_unix = 111
- integer, public commondata::platform_running

  Global variable that shows what is the current platform. Should use the above platform constants, e.g. `Platform_Running = PLATFORM_WINDOWS`. See `commondata::platform_windows` and `commondata::platform_unix`.
- character(len= *), parameter, public commondata::exec_interpolate = "htintrpl.exe"

  *There are a few **external programs** which are called from the model code. The name of the **interpolation** program (htintrpl.f90 from HEDTOOLS) executable.*
- character(len= *), parameter, public commondata::exec_scatterplot = "htscatter.exe"

  *The name of the **scatterplot** program (htscatter.f90 from HEDTOOLS) executable.*
- character(len= *), parameter, public commondata::exec_histogram = "hthist.exe"

  *The name of the **histogram** program (hthist.f90 from HEDTOOLS) executable.*
- character(len= *), parameter, public commondata::ltag_major = "IMPORTANT: "

  ***Tag prefixes** for the logger system. The log may use tags for some common information pieces, so they are easily found within. The tags are normally set the prefix for the log: 017-01-31 13:33:22 INFO: Saving histogram, data: debug_hist.csv Some common tags are: STAGE STAGE: 2017-01-31 16:03:15 INFO: Generation 7 took 448.3279s. INFO INFO: some information TIMER TIMER: Calculating distances took 0.001 s Tag meaning:*

- character(len= *), parameter, public commondata::ltag_stage = "STAGE: "
- character(len= *), parameter, public commondata::ltag_info = "INFO: "
- character(len= *), parameter, public commondata::ltag_warn = "WARNING: "
- character(len= *), parameter, public commondata::ltag_error = "ERROR: "
- character(len= *), parameter, public commondata::ltag_crit = "CRITICAL: "
- character(len= *), parameter, public commondata::ltag_timer = "TIMER: "
- character(len= *), parameter, public commondata::ltag_stats = "STATS: "

System-wide fatal errors

The description of errors that pertain to the whole system.

- character(len= *), parameter, public commondata::error_no_autoalloc = "No automatic array allocation"

  *Error message for **"no automatic intrinsic array allocation"**. Fortran compilers support automatic allocation of arrays on intrinsic assignment. This feature should work by default in GNU gfortran v.4.6 and Intel ifort v.17.0.1. Automatic allocation allows to avoid a possible bug when the number of array elements in the `allocate` statement is not updated when the components of the array are updated in the array constructor.*
- character(len= *), parameter, public commondata::error_auto_param_arrays = "No automatic size in parameter arrays"

  *Error message for **"no automatic determination of the size in parameter"** arrays in the style:*
- character(len= *), parameter, public commondata::error_allocation_fail = "Cannot allocate array or object"

  *Error message **"Cannot allocate array or object"** is issued if an array or an object is checked and turns out to be not allocated while it must be.*
- character(len= *), parameter, public commondata::error_lock_preexists = "Lock file '" // LOCK_FILE // "' exists. Is another simulation running?"

General Parameters

- character(len= *), parameter, public commondata::model_name = "HEDG2_04"

  Model name for tags, file names etc. Must be very short. See Model descriptors.

- character(len= *), parameter, public commondata::model_descr = "AHA, single fear, body size non-genetic."

  Model description - a fixed descriptive text, used in text outputs etc. See Model descriptors.
- character(len= *), parameter, private commondata::model_abstract_file = "abstract.txt"

  The name of the file that contains the Model abstract, a short description that can span several lines of text ans is kept in a separate file. The file is read, if it exists, and its contents is logged at the start the simulation. The separate Model Abstract file is useful because it can integrate dynamic information, such as the latest version control log(s) via Subversion or Mercurial hooks mechanism. See Model descriptors.
- logical, public, protected commondata::is_debug =.FALSE.

  Sets the model in the debug mode if TRUE. The Debug mode generates huge additional outputs and logs. Also, the logs by default go to the screen (standard output). See `commondata::system_init()` for details.
- logical, public, protected commondata::is_plotting =.TRUE.

  This parameter controls if the debug plots are produced. They can be huge number that takes lots of space. Also, debug plots are called as separate processes that can run at the background and easily exceed the system-specific limit on child processes (if run in asynchronous mode). Generation of debug plots can be controlled by the environment variable `AHA_DEBUG_PLOTS`: if it is set to TRUE, 1, or YES, debug plots are enabled. See `commondata::system_init()` for details.
- logical, public, protected commondata::is_screen_output =.FALSE.

  Sets the model in screen output mode. If TRUE, the logger output goes to the screen (standard output device). Can be manipulated using the environment variable `AHA_SCREEN`. If `AHA_SCREEN` is set to TRUE or 1 or yes, logger screen output is enabled. See `commondata::system_init()` for details.
- logical, public, protected commondata::is_zip_outputs =.FALSE.

  This parameter enables or disables post-processing compression of the data: if TRUE, the data are compressed using the command defined by the commondata::cmd_zip_output string parameter. Note that not all data files are compressed, only potentially big ones are (e.g. agent population data and habitat data).
- logical, parameter, public commondata::zip_outputs_background =.TRUE.

  This parameter defines if the output files are compressed in the background in the parallel mode or the program should wait for termination of the child zipping process.
- character(len= *), parameter, public commondata::cmd_zip_output = "gzip"

  This parameter defines the compression program that is executed to "zip" the data files if commondata::is_zip_outputs is enabled (TRUE). The normal compression utility is "gzip," that is found on almost any Linux/Unix system. gzip compresses each file individually and by default automatically deletes the original file. The compressed file extension is defined by commondata::zip_file_extenssion. See `http://www.gzip.org/`. Alternative compressors that are fairly widespread are `bzip2`, `lzma` and `xz`.
- character(len= *), parameter, public commondata::zip_file_extenssion = ".gz"

  This parameter defines the compressed file extension for the external compression utility defined by the commondata::cmd_zip_output.
- logical, parameter, public commondata::enable_save_agents_each_timestep = .FALSE.

  This parameter defines if all agents data is saved at each time step of the life cycle. See the_evolution::lifecycle_preevol().
- character(len=:), allocatable, public, protected commondata::mmdd

  `MMDD` tag, year, month and day, used in file names and outputs. The value of the tag should be obtained only once at the start of the simulation, normally by calling the commondata::tag_mmdd() function at commondata::system_init(). It does not make much sense to generate these data tags on the fly as the simulations can be very long, several days, and so the file tags will be inconsistent.
- integer, parameter, public commondata::popsize = 10000

  Maximum population size.
- integer, parameter, public commondata::generations = 100

  Maximum number of generations in GA.
- integer, public commondata::global_generation_number_current

  *The current global **generation number**. This is a global non fixed-parameter variable that is updated in subroutines.*
- integer, parameter, public commondata::lifespan = 14000

  Number of time steps in the agent's maximum life length.
- integer, parameter, public commondata::preevol_tsteps = 560

  Number of time steps in the agent's life at the pre-evolution stage.
- integer, parameter, public commondata::preevol_tsteps_force_debug = 280

  *Number of time steps in the agent's life at the fixed fitness pre-evolution stage. This parameter **forces** a smaller fixed value that is used for debugging only. Thus, adaptive time steps calculated by the_evolution::preevol_steps_adaptive() are disabled. To enable this fixed time steps, set this parameter commondata::preevol_tsteps_force_debug_enabled to TRUE.*

- logical, parameter, public commondata::preevol_tsteps_force_debug_enabled = .FALSE.

  This parameter enables the forced smaller fixed number of time steps set by the commondata::preevol_tsteps_force_debug parameter.
- logical, parameter, public commondata::lifecycle_predation_disabled_debug = .FALSE.

  This parameter completely disables predation in the GA life cycle procedure.
- integer, public commondata::global_time_step_model_current

  *The current global **time step** of the model. This is a global non fixed-parameter variable that is updated in subroutines.*
- integer, public commondata::global_frame_number

  The current global time frame. Frames are time steps within the time step defined by the commondata::global_time_step_model_curren
- real(srp), parameter, public commondata::percept_error_cv_def = 0.01_SRP

  Default perception error in the commondata::gamma2gene() neuronal response functions. Note that this parameter defines stochastic error as the Coefficient of Variation (CV).

Basic agent parameters

- real(srp), parameter, public commondata::body_length_min = 0.2_SRP

  Minimum body length possible.
- real(srp), parameter, public commondata::body_length_max = 100.0_SRP

  Maximum body length.
- real(srp), parameter, public commondata::body_mass_min = 0.1_SRP

  Minimum possible body mass, hard limit.
- logical, parameter, public commondata::init_agents_depth_is_fixed = .FALSE.

  This parameter determines if the agents are initialised at a fixed depth at the initialisation. Agents are normally placed uniformly, the_environment::uniform(), at the initialisation. However, the depth can be fixed. In such a case they are scattered uniformly in the X and Y coordinates but with fixed depth that is set by the commondata::init_agents_depth parameter.
- logical, parameter, public commondata::init_agents_depth_is_gauss = .TRUE.

  This parameter determines if the agents are initialised at a fixed depth at the initialisation. Agents are placed uniformly, the_environment::uniform(), at the initialisation. However, the depth can be a Gaussian value with the.
- real(srp), parameter, public commondata::init_agents_depth = 1833.0_SRP

  The fixed depth at which the agents are initialised at the start of the simulation. The other coordinates are normally set the_environment::uniform() within the initialisation environment container. See the_population::member_population::place_uniform().
- real(srp), parameter, public commondata::init_agents_depth_cv = 0.2_SRP

  This parameter sets the Coefficient of Variation for the Gaussian depth initialisation of the agents that is controlled by commondata::init_agents_depth_is_gauss. See the_population::member_population::place_uniform().
- real(srp), parameter, public commondata::reproduction_cost_body_mass_fix = 0.2_SRP

  The energetic cost of reproduction in terms of the agent's body mass loss.
- real(srp), parameter, public commondata::reproduction_cost_offspring_fract_male = 0.3_SRP

  *The component of the energetic cost of reproduction in **males** that is proportional to the total offspring mass. For details see the procedure `the_body::reproduction_cost_energy_dynamic()`.*
- real(srp), parameter, public commondata::reproduction_cost_offspring_fract_female = 1.0_SRP

  *The component of the energetic cost of reproduction in **females** that is proportional to the total offspring mass. For details see the procedure `the_body::reproduction_cost_energy_dynamic()`.*
- real(srp), parameter, public commondata::reproduction_cost_body_mass_factor_male = 0.4_SRP

  *The component of the energetic cost of reproduction in **males** that is proportional to the agent's body mass. For details see the procedure `the_body::reproduction_cost_energy_dynamic()`.*
- real(srp), parameter, public commondata::reproduction_cost_body_mass_factor_female = 0.1

  *The component of the energetic cost of reproduction in **females** that is proportional to the agent's body mass. For details see the procedure `the_body::reproduction_cost_energy_dynamic()`.*
- real(srp), parameter, public commondata::reproduction_cost_unsuccess = 0.1_SRP

  *The energetic cost of unsuccessful reproduction in terms of the agent's body mass lost. This is a fraction of the **full cost of reproduction**, that is described by the `REPRODUCTION_COST_BODY_MASS` parameter.*
- real(srp), dimension(∗), parameter, public commondata::reproduct_body_mass_offspr_abscissa = [BODY_MASS_MIN, 3.0_SRP, 10.5_SRP, 12.0_SRP]

  *The array defining the **abscissa** (X) of the nonparametric function curve that defines the relationship between the agent's body mass and the overall mass of all offspring as a fraction of the agent's body mass.*
- real(srp), dimension(∗), parameter, public commondata::reproduct_body_mass_offspr_ordinate = [0.0_↩ SRP, 0.1_SRP, 0.199_SRP, 0.20_SRP]

*The array defining the **ordinate** (Y) of the nonparametric function curve that defines the relationship between the agent's body mass and the overall mass of all offspring as a fraction of the agent's body mass. Plotting command for the interpolator:*

Parameters of the environment

- real(srp), dimension(3), parameter, public commondata::universe_min_coord_notuse = [0.0_SRP, 0.0_↩ SRP, 0.0_SRP]

  Overall size of the global 3D universe of the model.
- real(srp), dimension(3), parameter, public commondata::universe_whole_size_notuse = [20000.0_SRP, 10000.0_SRP, 3000.0_SRP]
- integer, parameter, public commondata::dielcycles = 100

  Number of days and nights in a lifespan, `DIELCYCLES=500`.
- integer, parameter, public commondata::history_size_spatial = 50

  The size of the history for spatial moving objects, i.e. how many time steps positions to remember in stack arrays.
- real(srp), dimension(3), parameter, public commondata::habitat_safe_min_coord = [0.0_SRP, 0.0_SRP, 0.0_SRP]

  Definition of the habitat spatial limits.
- real(srp), dimension(3), parameter, public commondata::habitat_safe_max_coord = [10000.0_SRP, 10000.0_SRP, 3000.0_SRP]
- real(srp), dimension(3), parameter, public commondata::habitat_danger_min_coord = [10000.0_SRP, 0.↩ 0_SRP, 0.0_SRP]
- real(srp), dimension(3), parameter, public commondata::habitat_danger_max_coord = [20000.0_SRP, 10000.0_SRP, 3000.0_SRP]
- integer, parameter, public commondata::predators_num_habitat_safe = 100

  *The **number of predators** in the **safe** habitat.*
- integer, parameter, public commondata::predators_num_habitat_danger = 500

  *The **number of predators** in the **dangerous** habitat.*
- integer, parameter, public commondata::food_abundance_habitat_safe = 20000

  *The **food abundance** in the **safe** habitat.*
- integer, parameter, public commondata::food_abundance_habitat_danger = 40000

  *The **food abundance** in the **dangerous** habitat.*
- real(srp), parameter, public commondata::other_risks_def = 0.01_SRP

  Default level of other mortality risks in the habitat.
- real(srp), parameter, public commondata::other_risks_habitat_safe = 0.01_SRP

  *Habitat-specific mortality risk (not linked with predation) in the **safe** habitat.*
- real(srp), parameter, public commondata::other_risks_habitat_danger = 0.05_SRP

  *Habitat-specific mortality risk (not linked with predation) in the **dangerous** habitat.*
- real(srp), parameter, public commondata::eggmortality_def = 0.01_SRP

  Default level of egg mortality in the habitat.
- real(srp), parameter, public commondata::individual_mortality_risk_def = 0.01_SRP

  Default individually-specific mortality risk. It can increase or decrease depending on various factors. The individually-specific mortality risk is normally a Gaussian variable with the variability set by the commondata::individual_mortality_risk_cv.
- real(srp), parameter, public commondata::individual_mortality_risk_cv = 0.05_SRP

  The coefficient of variation for Gaussian stochastic individually-specific mortality risk of the agent.
- real(srp), parameter, public commondata::predator_body_size = 100.0_SRP

  The body size of the predator. In this version all predators have the same body size set by this parameter, but can be Gaussian stochastic. Moreover, in such a case predator attack efficiency can depend on the body size, e.g. larger predators are more dangerous. compare to the agents maximum body size `BODY_LENGTH_MAX=100.0`
- real(srp), parameter, public commondata::predator_attack_rate_default = 0.9_SRP

  Mean rate of a single predator attack.
- real(srp), parameter, public commondata::predator_attack_rate_cv = 0.1_SRP

  Coefficient of variation for a single predator attack among the whole population of stochastic predators.
- real(srp), parameter, public commondata::predator_attack_capture_probability_half = 0.8_SRP

  The probability of capture of a fish agent by a predator at the distance equal to 1/2 of the visual range. For more details see `the_environment::predator_capture_risk_calculate_fish()`.
- real(srp), parameter, public commondata::predator_attack_capture_probability_min = 0.1_SRP

  Minimum probability of capture, e.g. at a distance exceeding the visual range. The latter assumes that the predator could detect the agent beyond the visual range and pursue it. For more details see `the_environment::predator_capture_risk_calculate_fish()`.

- real(srp), parameter, public commondata::predator_attack_capture_prob_frz_50 = 0.10_SRP

  *A parameter factor defining the probability of capture of an immobile (freezing) agent by a predator: interpolation ordinate for the distance equal to **0.25 of the visual range**. See `the_environment::predator_capture_risk_calculate` for details.*
- real(srp), parameter, public commondata::predator_attack_capture_prob_frz_75 = 0.01_SRP

  *A parameter factor defining the probability of capture of an immobile (freezing) agent by a predator: interpolation ordinate for the distance equal to **0.40 of the visual range**. See `the_environment::predator_capture_risk_calculate` for details.*
- logical, parameter, public commondata::agent_can_assess_predator_attack_rate = .TRUE.

  A logical flag of whether the agents can assess the individual inherent attack rates of the predators. If yes, these inherent individual attack rates are collated into the perception object. If no, the default attack rate set by the commondata::predator_attack_rate_default parameter is used.
- integer, parameter, public commondata::predator_risk_group_select_index_partial = 20

  *Sets the limit for partial indexing and ranking of **prey agents** in the visual range of the predator. The risk of predation, i.e. the probability of attack and capture of each agent in a group of agents, will be calculated individually for distance-ranked agents only up to this parameter value.*
- real(srp), dimension(∗), parameter, public commondata::predator_risk_group_dilution_ordinate = [1.0_↩ SRP, 0.3_SRP, 0.1_SRP]

  The array defining the ordinate grid values for the weighting nonparametric function linking the distance rank of the agent within the visual field of the predator and the weighting factor adjusting for predator confusion and predator dilution effects. The grid abscissa is calculated dynamically in the the_environment::predator_capture_risk_calculate_fish_group() procedure.
- real(srp), parameter, public commondata::food_item_size_default = 2.1_SRP

  Default size of a single food item.
- real(srp), parameter, public commondata::food_item_mean_size = FOOD_ITEM_SIZE_DEFAULT

  The above is also the average size of a stochastic Gaussian food items.
- real(srp), parameter, public commondata::food_item_size_default_cv = 0.1_SRP

  Coefficient of variation for Gaussian food items.
- real(srp), parameter, public commondata::food_item_minimum_size = 1.0_SRP

  The minimum size of a food item. This is the "floor" in case the stochastically generated (e.g. Gaussian) value gets zero or below.
- real(srp), parameter, public commondata::food_item_density = 0.1_SRP

  The (physical) density of a single food item. TODO: need to parametrise!
- real(srp), parameter, public commondata::food_item_capture_prop_cost = 0.05_SRP

  *The cost of the food item catching, in terms of the **food item mass** (proportional cost). So, if the agent does an unsuccessful attempt to catch a food item, the cost still applies.*
- real(srp), parameter, public commondata::food_item_capture_probability = 0.99_SRP

  *The **baseline** probability that the food item is captured. See the_neurobio::food_item_capture_probability_calc().*
- real(srp), parameter, public commondata::food_item_capture_probability_min = 0.1_SRP

  *The **minimum** probability of capture a food item, when the item is at a distance equal to the visual range from the predator agent.*
- real(srp), parameter, public commondata::food_item_capture_probability_subjective_errorr_cv = 0.1

  Subjective error assessing the food item capture probability when assessing the subjective GOS expectancies of food items. The subjective assessment value of the capture probability is equal to the objective value plus random error with the CV equal to this parameter.
- real(srp), parameter, public commondata::food_item_migrate_xy_mean = FOOD_ITEM_SIZE_DEFAULT ∗ 10.0_SRP

  Mean shift parameter for the local random walk movement of food items in the horizontal plane.
- real(srp), parameter, public commondata::food_item_migrate_depth_mean = FOOD_ITEM_SIZE_↩ DEFAULT ∗ 100.0_SRP

  Mean shift parameter for the local random walk movement of food items in the vertical (depth) plane.
- real(srp), parameter, public commondata::food_item_migrate_xy_cv = FOOD_ITEM_SIZE_DEFAULT_CV

  Coefficient of variation parameter for the local random walk movement of food items in the horizontal plane.
- real(srp), parameter, public commondata::food_item_migrate_depth_cv = 0.8_SRP

  Coefficient of variation parameter for the local random walk movement of food items in the vertical (depth) plane.
- real(srp), parameter, public commondata::daylight =500.0_SRP

  Maximum above-surface light intensity at midday, DAYLIGHT=500.0.
- logical, parameter, public commondata::daylight_stochastic = .TRUE.

  Flag for stochastic daylight pattern (if TRUE) or deterministic sinusoidal (when FALSE). Check out the next parameter `DAYLIGHT_CV` for variability.

- real(srp), parameter, public commondata::daylight_cv =0.2_SRP
  Coefficient of variation for stochastic DAYLIGHT,.
- real(srp), parameter, public commondata::beamatt =1.0_SRP
  Beam attenuation coefficient of water (m-1),BEAMATT = 1.0.
- real(srp), parameter, public commondata::preycontrast_default = 1.0_SRP
  Inherent contrast of prey, CONTRAST =1.0.
- real(srp), parameter, public commondata::preyarea_default =3.E-6_SRP
  Area of prey (m2), PREYAREA = 3.E-6.
- real(srp), parameter, public commondata::viscap =1.6E6_SRP
  Dimensionless descriptor of fish eye quality, VISCAP=1.6E6.
- real(srp), parameter, public commondata::eyesat =500.0_SRP
  Saturation parameter of eye (Ke) (uE m-2 s-1), EYESAT=500.0.
- real(srp), parameter, public commondata::lightdecay = 0.002_SRP
  Vertical conservation of light, per depth (old code lightdecay=0.2).

Genetic architecture parameters

- integer, parameter, public commondata::allelerange_min = 1
  The minimum possible value of alleles (allele range minimum) See implementation notes on the_genome::gene::allele_val component of the the_genome::gene derived type and commondata::alleleconv() and commondata::allelescale() functions.
- integer, parameter, public commondata::allelerange_max = 10000
  The maximum possible value of alleles (allele range maximum) See implementation notes on the_genome::gene::allele_va component of the the_genome::gene derived type and commondata::alleleconv() and commondata::allelescale() functions.
- real(srp), parameter, public commondata::allelescale_max = 20.0_SRP
  Conversion parameter that defines the scaling of the integer allele values ::ALLELERANGE_MIN to ALLELERANGE_MAX are converted to zero to this parameter value as the maximum. See allelescale() for details.
- integer, parameter, public commondata::additive_comps = 3
  Number of additive allele components.
- real(srp), parameter, public commondata::mutationrate_point = 0.1_SRP
  Mutation rate for point allele mutations.
- real(srp), parameter, public commondata::ga_mutationrate_point_max = 0.25_SRP
  Maximum point mutation rate in the adaptive Fixed Fitness Genetic Algorithm.
- real(srp), parameter, public commondata::mutationrate_batch = 0.05_SRP
  Mutation rate for point allele mutations, a whole batch of allele components.
- real(srp), parameter, public commondata::ga_mutationrate_batch_max = 0.1_SRP
  Maximum batch mutation rate in the adaptive Fixed Fitness Genetic Algorithm.
- real(srp), parameter, public commondata::relocation_swap_rate = 0.05_SRP
  Mutation rate for chromosome relocation, i.e. probability of a gene moving to a different position on the same chromosome: There are two kinds of relocations, swapping genes between two positions and moving a gene with subsequent shift. So we have two constants for the respective rates.
- real(srp), parameter, public commondata::relocation_shift_rate = 0.01_SRP
- integer, parameter, public commondata::n_chromosomes = 6
  The number of chromosomes for the agents.
- integer, dimension(n_chromosomes), parameter, public commondata::len_chromosomes = [6, 5, 12, 12, 12, 12]
  The number of alleles in each of the chromosomes. NOTE: This must be an array (vector) of the size commondata::n_chromosomes. We use new Fortran array constructor here to set the array values.
- integer, parameter, public commondata::max_nalleles = 12
  This parameter defines the maximum number of alleles within the chromosome It IS NOT intended to vary freely/independently. Used in definitions of _GENOTYPE_PHENOTYPE matrices, equal to the maxval(LEN↩ _CHROMOSOMES).
- character(len= ∗), dimension(n_chromosomes), parameter, public commondata::lab_chromosomes = ["C_1_SEX ", "C_2_BODY", "C_3_HORM", "C_4_HUNG", "C_5_FEAR", "C_6_REPR"]
  Set the labels of the chromosomes. NOTE, must be an array(vector)) of the size commondata::n_chromosomes. We use new Fortran array constructor here to set the array values.
- integer, parameter, public commondata::chromosome_ploidy = 2
  The ploidy of the chromosome set. Can theoretically be haploid (=1), diploid (=2) or, polyploid (>2).

- real(srp), parameter, public commondata::genome_recombination_ratio_mother = 0.8_SRP

  *The ratio of the genome that inherited from the **mother**. The other part is inherited from the father. See implementation details of the **random independent** genetic recombination procedure `procedures the_genome::individual_genome::recombine()`.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::genome_crossover_fixed_mother = reshape ([YES,YES,YES,YES,YES,YES, YES,YES,YES,YES,YES,YES, YES,YES,YES,YES,YES,YES, NO, NO,YES,YES,YES,YES, NO, NO,YES,YES,YES,YES, NO, NO,YES,YES,YES,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *Boolean 2D matrix that determines the pattern of **fixed** chromosome **crossover**. For each chromosome, the alleles that are marked with the TRUE (YES) values are inherited from the **mother** whereas those marked FALSE (NO) are inherited from the **father**. See implementation details of the **fixed genetic crossover** procedure `procedures the_genome::individual_genome::crossover()`.*
- real(srp), parameter, public commondata::sex_ratio = 0.5_SRP

  Sex ratio for initialising genomes.
- character(len=label_length), parameter commondata::sexlocus_label ="SEX_LOCUS"

  Labels for the sex locus alleles (gene) - vector as we don't need to label individual alleles. `LABEL_LENGTH` is here to avoid a GF warning.
- character(len= ∗), parameter, public commondata::male ="male"

  Set names of the sexes – the allele labels.
- character(len= ∗), parameter, public commondata::female ="female"
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::sex_genotype_phenotype = reshape ([YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *Sex definition can be implemented differently from all other traits. Here is an example of the **phenotype** x **genotype matrix** code for sex determination:*

Hormonal parameters

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::growhorm_genotype_phenotype = reshape ([NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *Genotype x Phenotype matrix for **growth hormone**.*
- real(srp), parameter, public commondata::growhorm_init = 0.5_SRP

  *Genotype to phenotype gamma2gene initialisation value for **growth hormone***
- real(srp), parameter, public commondata::growhorm_gerror_cv = 0.5_SRP

  Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::thyroid_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *Genotype x Phenotype matrix for **thyroid**.*
- real(srp), parameter, public commondata::thyroid_init = 0.5_SRP

  *Genotype to phenotype gamma2gene initialisation value for **thyroid***
- real(srp), parameter, public commondata::thyroid_gerror_cv = 0.5_SRP

  Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::adrenaline_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *Genotype x Phenotype matrix for **adrenaline***
- real(srp), parameter, public commondata::adrenaline_init = 0.5_SRP

*Genotype to phenotype gamma2gene initialisation value for **adrenaline***

- real(srp), parameter, public commondata::adrenaline_gerror_cv = 0.5_SRP

  Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::cortisol_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *Genotype x Phenotype matrix for **cortisol**.*

- real(srp), parameter, public commondata::cortisol_init = 0.5_SRP

  *Genotype to phenotype gamma2gene initialisation value for **cortisol***

- real(srp), parameter, public commondata::cortisol_gerror_cv = 0.5_SRP

  Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::testosterone_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *Genotype x Phenotype matrix for **testosterone**.*

- real(srp), parameter, public commondata::testosterone_init = 0.01_SRP

  *Genotype to phenotype gamma2gene initialisation value for **testosterone***

- real(srp), parameter, public commondata::testosterone_gerror_cv = 0.5_SRP

  Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::estrogen_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  Genotype x Phenotype matrix for ESTROGEN.

- real(srp), parameter, public commondata::estrogen_init = 0.01_SRP

  *Genotype to phenotype gamma2gene initialisation value for **estrogen***

- real(srp), parameter, public commondata::estrogen_gerror_cv = 0.5_SRP

  Genotype to phenotype gamma2gene Gaussian error parameter. This is really the coefficient of variation of the output hormone level with respect to an ideal value (initially 0).

- integer, parameter, public commondata::sex_steroids_check_history = 3

  *The number of the latest historical values that are checked for change when setting an increment of the **sex steroid** hormones.*

- real(srp), dimension(∗), parameter, public commondata::sex_steroids_increment_factor_age_curve_abscissa = [0.0_SRP, real(LIFESPAN, SRP)∗0.25_SRP, real(LIFESPAN, SRP)]

  *The array defining the **abscissa** (X) of the nonparametric function curve that defines the relationship between the age of the agent and the steroid increment factor for this specific age.*

- real(srp), dimension(∗), parameter, public commondata::sex_steroids_increment_factor_age_curve_ordinate = [0.0_SRP, 0.01_SRP, 0.1_SRP]

  *The array defining the **ordinate** (Y) of the nonparametric function curve that defines the relationship between the age of the agent and the steroid increment factor for this specific age.*

- real(srp), dimension(∗), parameter, public commondata::sex_steroids_increment_factor_len_curve_abscissa = [0.0_SRP, BODY_LENGTH_MAX∗0.2_SRP, BODY_LENGTH_MAX]

  *The array defining the **abscissa** (X) of the nonparametric function curve that defines the relationship between the body length of the agent and the steroid increment factor for this specific length.*

- real(srp), dimension(∗), parameter, public commondata::sex_steroids_increment_factor_len_curve_ordinate = [0.0_SRP, 0.01_SRP, 0.1_SRP]

  *The array defining the **ordinate** (Y) of the nonparametric function curve that defines the relationship between the body length of the agent and the steroid increment factor for this specific length.*

Body parameters, individual physiology and condition

- integer, parameter, public commondata::history_size_agent_prop = 100

History stack size for the agent's basic properties, such as body length and body mass. Normally they are saved only for the analysis and currently not used in the perception.

- real(srp), parameter, public commondata::living_cost = 4.0_SRP

  Living cost in terms of food consumed. metabolic costs, p roportional to body size.
- real(srp), parameter, public commondata::mass_growth_threshold = 0.0001_SRP

  A minimum body mass increment when any linear growth is possible, in units of the body mass (e.g. 0.05 = 5%)
- real(srp), parameter, public commondata::linear_growth_exponent = 3.0_SRP

  Growth exponent linking linear growth and body mass growth. Based on Fulton's condition factor "cube law.".
- real(srp), dimension(∗), parameter, public commondata::linear_growth_hormone_increment_factor_curve_abscissa = [0.0_SRP, GROWHORM_INIT, GROWHORM_INIT∗3.0_SRP, GROWHORM_INIT∗5.0_SRP, GROWHORM_INIT∗20.0_SRP]

  *The array defining the **abscissa** (X) of the nonparametric function curve that defines the function linking the relationship between the growth hormone and the relative linear growth increment.*
- real(srp), dimension(∗), parameter, public commondata::linear_growth_hormone_increment_factor_curve_ordinate = [0.0_SRP, 0.6_SRP, 0.9_SRP, 0.98, 1.00_SRP]

  *The array defining the **ordinate** (Y) of the nonparametric function curve that defines the function linking the relationship between the growth hormone and the relative linear growth increment.*
- real(srp), parameter, public commondata::max_stomach_capacity_def = 0.15_SRP

  Set the maximum stomach capacity default value – fraction of the body mass available for food. Can be overriden in different agent types. Normally 15%.
- real(srp), parameter, public commondata::stomach_content_emptify_factor = 100.0_SRP

  Stomach content emptify factor at each step.
- real(srp), parameter, public commondata::stomach_content_init = 0.01_SRP

  Set average stomach capacity at birth/init in units of body weight,.
- real(srp), parameter, public commondata::stomach_content_init_cv = 0.05_SRP

  Set the coefficient of variation for the stomach capacity at init.
- real(srp), parameter, public commondata::swimming_speed_cost_burst = 0.1E-4_SRP

  Set the weighting factor parameter of burst swimming cost in terms of the agent body size and the distance expressed in terms of the agent body lengths. In the_body::condition_cost_swimming_burst(), this parameter sets the β coefficient.
- real(srp), parameter, public commondata::cost_factor_foraging_smr = 0.1E-5_SRP

  Set the cost of foraging in terms of SMR.
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::energy_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The initial value of the **energy reserves** at birth is genetically determined. This is the Genotype x Phenotype matrix for energy reserves.*
- real(srp), parameter, public commondata::energy_init = 0.18

  This is the initial value of the energy reserves, non-genetic mean.
- real(srp), parameter, public commondata::energy_gerror_cv = 0.01

  Genotype to phenotype initialisation, Gaussian error parameter. Coefficient of variation for the $ENERGY\_INIT$ value.
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::body_length_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  The initial value of the body length at birth is genetically determined. This is the Genotype x Phenotype matrix for body length.
- real(srp), parameter, public commondata::body_length_init = 2.3_SRP

  *The initial value of **body length**, the average (gon-genetic).*
- real(srp), parameter, public commondata::body_length_gerror_cv = 0.1_SRP

  Genotype to phenotype initialisation, Gaussian error parameter. Coefficient of variation for the $BODY\_LENGTH\hookleftarrow\_INIT$ value.
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::control_unselected_genotype_phenotyp = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

*The initial value of the **control unselected** trait. This trait is is genetically determined but is not selected or used. So it can be used to control for random genetic drift. This is the Genotype x Phenotype matrix.*

- real(srp), parameter, public commondata::control_unselected_init = 0.5_SRP

  *The initial value of the **control unselected** trait that goes through the gamma2gene.*
- real(srp), parameter, public commondata::control_unselected_gerror_cv = 0.5_SRP

  Genotype to phenotype initialisation, Gaussian error parameter. Coefficient of variation for the control unselected trait.
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::smr_genotype_phenotype = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  The initial value of the standard metabolic rate (SMR) at birth is genetically determined. This is the Genotype x Phenotype matrix for SMR.
- real(srp), parameter, public commondata::smr_init = 0.5_SRP

  This is the initial value of SMR that goes through the gamma2gene.
- real(srp), parameter, public commondata::smr_gerror_cv = 0.5_SRP

  Genotype to phenotype initialisation, Gaussian error parameter. Coefficient of variation for the `SMR_LENGTH_INIT` value.
- real(srp), parameter, public commondata::smr_min = 0.01_SRP

  Minimum SMR value, anything lower is not allowed.
- real(srp), parameter commondata::swimming_cost_exponent_laminar = 0.5_SRP

  *Default swimming cost body mass exponent parameter for **laminar** flow. See doi:10.1242/jeb.01484 (`https://dx.doi.org/10.1242/jeb.01484`) and the_body::condition_cost_swimming_burst() for details.*
- real(srp), parameter commondata::swimming_cost_exponent_turbulent = 0.6_SRP

  *Default swimming cost body mass exponent parameter for **turbulent** flow. See doi:10.1242/jeb.01484 (`https://dx.doi.org/10.1242/jeb.01484`) and the_body::condition_cost_swimming_burst() for details.*
- real(srp), parameter, public commondata::swimming_cost_factor_buoyancy_down = 0.01_SRP

  This parameter defines the cost of the buoyancy-based locomotion as a fraction of normal laminar flow propulsion for lowering downwards.
- real(srp), parameter, public commondata::swimming_cost_factor_buoyancy_up = 0.1_SRP

  This parameter defines the cost of the buoyancy-based locomotion as a fraction of normal laminar flow propulsion for lowering downwards.

Parameters of the neurobiological architecture

- integer, parameter, public commondata::food_select_items_index_partial = 20

  *Sets the limit for partial indexing and ranking of **food items** in the visual range of the agents.*
- integer, parameter, public commondata::consp_select_items_index_partial = 20

  *Sets the limit for partial indexing and ranking of **conspecifics** in the visual range of the agent.*
- integer, parameter, public commondata::pred_select_items_index_partial = 20

  *Sets the limit for partial indexing and ranking of **predators** in the visual range of the agent.*
- real(srp), parameter, public commondata::individual_visual_contrast_default = 1.0_SRP

  *Inherent contrast of the **agent**, It is used in determining the visual range of an agent in perception of conspecifics, and also for assessing the agent's detectability by predator. Default Contrast of food items is set separately.*
- integer, parameter, public commondata::history_size_perception = HISTORY_SIZE_SPATIAL

  Sets the size of the perception memory stack.
- integer, parameter, public commondata::history_size_motivation = HISTORY_SIZE_SPATIAL

  Sets the size of the emotional state memory stack.
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::light_hunger_genotype_neuronal = reshape ([NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **light** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::light_hunger_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **light** perception effects on **hunger**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::depth_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **depth** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::depth_hunger_genotype_neuronal_gerror_cv = PERCEPT_↩ ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **depth** perception effects on **hunger**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::foodcount_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **food items count** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::foodcount_hunger_genotype_neuronal_gerror_cv = PERCEPT↩ _ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **food items count** perception effects on **hunger**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::food_mem_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **food items count** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::food_mem_hunger_genotype_neuronal_gerror_cv = PERCEPT↩ _ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **food items count** perception effects on **hunger**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::conspcount_hunger_genotype_neurona = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **conspecifics number** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::conspcount_hunger_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **conspecifics number count** perception effects on **hunger**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::pred_direct_hunger_genotype_neurona = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **direct predation** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::pred_direct_hunger_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **direct predation** perception effects on **hunger**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::pred_meancount_hunger_genotype_ne = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **mean predator number** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::pred_meancount_hunger_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

*Gaussian perception error parameter (cv) for **mean predator number** perception effects on **hunger**.*

- real(srp), parameter, public commondata::predation_risk_weight_immediate = 0.7

  The weight of the immediately seen predators over those in the perceptual memory stack. Those in the memory then have this weight in the predation risk estimate: `1-PREDATION_RISK_WEIGHT_IMMEDIATE`. See the_neurobio::predation_risk_backend().

- integer, parameter, public commondata::predation_risk_weight_memory_window = nint(HISTORY_SIZE←_PERCEPTION / 2.0)

  The length of the perceptual memory window that is taken into account and weighted against the currently seen number of predators when calculating the motivation value.

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::stom_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **stomach** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::stom_hunger_genotype_neuronal_gerror_cv = PERCEPT_←ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **stomach** perception effects on **hunger**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::bodymass_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **body mass** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::bodymass_hunger_genotype_neuronal_gerror_cv = PERCEPT←_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **body mass** perception effects on **hunger**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::energy_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **energy reserves** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::energy_hunger_genotype_neuronal_gerror_cv = PERCEPT_←ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **energy reserves** perception effects on **hunger**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::age_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **age** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::age_hunger_genotype_neuronal_gerror_cv = PERCEPT_←ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **age** perception effects on **hunger**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::reprfac_hunger_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **reproductive factor** perception effects on **hunger** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::reprfac_hunger_genotype_neuronal_gerror_cv = PERCEPT_←ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **reproductive factor** perception effects on **hunger**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::light_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **light** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::light_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT←_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **light** perception effects on **fear state**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::depth_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **depth** perception effects on **active avoidance** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::depth_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT←_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **depth** perception effects on **fear state**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::foodcount_actv_avoid_genotype_neuro = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **food items count** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::foodcount_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **food items count** perception effects on **fear state**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::food_mem_actv_avoid_genotype_neuro = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **food items count** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::food_mem_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **food items count** perception effects on **fear state**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::conspcount_actv_avoid_genotype_neur = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **conspecifics number** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::conspcount_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **conspecifics number count** perception effects on **fear state**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::pred_direct_actv_avoid_genotype_neur = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **direct predation** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::pred_direct_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

*Gaussian perception error parameter (cv) for **direct predation** perception effects on **fear state**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::pred_meancount_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

    *The genotype structure for **mean predator number** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::pred_meancount_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

    *Gaussian perception error parameter (cv) for **mean predator number** perception effects on **fear state**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::stom_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

    *The genotype structure for **stomach** perception effects on **active avoidance** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::stom_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

    *Gaussian perception error parameter (cv) for **stomach** perception effects on **fear state**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::bodymass_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

    *The genotype structure for **body mass** perception effects on **active avoidance** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::bodymass_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

    *Gaussian perception error parameter (cv) for **body mass** perception effects on **fear state**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::energy_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

    *The genotype structure for **energy reserves** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::energy_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

    *Gaussian perception error parameter (cv) for **energy reserves** perception effects on **fear state**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::age_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

    *The genotype structure for **age** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::age_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

    *Gaussian perception error parameter (cv) for **age** perception effects on **fear state**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::reprfac_actv_avoid_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

    *The genotype structure for **reproductive factor** perception effects on **fear state** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::reprfac_actv_avoid_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

*Gaussian perception error parameter (cv) for **reproductive factor** perception effects on **fear state**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::light_reproduce_genotype_neuronal = reshape ([NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **light** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::light_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **light** perception effects on **reproduction**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::depth_reproduce_genotype_neuronal = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **depth** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::depth_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **depth** perception effects on **reproduction**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::foodcount_reproduce_genotype_neuron = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **food items count** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::foodcount_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **food items count** perception effects on **reproduction**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::food_mem_reproduce_genotype_neuro = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **food items count** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::food_mem_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **food items count** perception effects on **reproduction**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::conspcount_reproduce_genotype_neuro = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **conspecifics number** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::conspcount_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **conspecifics number count** perception effects on **reproduction**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::pred_direct_reproduce_genotype_neuro = reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **direct predation** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*
- real(srp), parameter, public commondata::pred_direct_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

*Gaussian perception error parameter (cv) for **direct predation** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::pred_meancount_reproduce_genotype_
= reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **mean predator number** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::pred_meancount_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **mean predator number** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::stom_reproduce_genotype_neuronal
= reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **stomach** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::stom_reproduce_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **stomach** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::bodymass_reproduce_genotype_neuro
= reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **body mass** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::bodymass_reproduce_genotype_neuronal_gerror_cv = PERCEPT_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **body mass** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::energy_reproduce_genotype_neuronal
= reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **energy reserves** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::energy_reproduce_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **energy reserves** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::age_reproduce_genotype_neuronal
= reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **age** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::age_reproduce_genotype_neuronal_gerror_cv = PERCEPT_↩
ERROR_CV_DEF

  *Gaussian perception error parameter (cv) for **age** perception effects on **reproduction**.*

- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::reprfac_reproduce_genotype_neuronal
= reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO,YES], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for **age** perception effects on **reproduction** that goes via gamma2gene perception to neuronal response.*

- real(srp), parameter, public commondata::reprfac_reproduce_genotype_neuronal_gerror_cv = PERCEPT↩
_ERROR_CV_DEF

*Gaussian perception error parameter (cv) for **age** perception effects on **reproduction**.*

- logical, parameter, public commondata::modulation_appraisal_disable_all = .FALSE.

  *Disable developmental or genetic modulation of the `APPRAISAL` completely. If set to TRUE, **no modulation** is performed and the final motivation values are just equal to the primary modulation values in `APPRAISAL`.*
- real(srp), parameter, public commondata::reprod_modulation_devel_agestart = LIFESPAN / 2.0_SRP

  Developmental modulation of reproductive motivation. This parameter defines the starting age (time steps) at which reproductive motivation can rise above zero (at all previous ages weight=0.0). See the_neurobio::appraisal_motivation_modulation_non_genetic().
- real(srp), parameter, public commondata::reprod_modulation_devel_agefull = REPROD_MODULATION↩
_DEVEL_AGESTART + REPROD_MODULATION_DEVEL_AGESTART ∗ 2.0_SRP / 3.0_SRP

  Developmental modulation of reproductive motivation. This parameter defines the age (time steps) at which reproductive motivation is weighted fully (weight = 1.0). See the_neurobio::appraisal_motivation_modulation_non_genetic().
- real(srp), dimension(∗), parameter commondata::reprod_modulation_devel_abscissa = [REPROD↩
_MODULATION_DEVEL_AGESTART, REPROD_MODULATION_DEVEL_AGESTART + (REPROD↩
_MODULATION_DEVEL_AGEFULL - REPROD_MODULATION_DEVEL_AGESTART) / 3.0_SRP, REPROD_MODULATION_DEVEL_AGEFULL]

  Developmental modulation of reproductive motivation. This is the interpolation grid abscissa defining at which age reproduction motivation can have non-zero values (reproduction is possible). It defines the weighting factor applied to reproductive motivation.
- real(srp), parameter, public commondata::reprod_modulation_devel_w2 = 0.1_SRP

  Developmental modulation of reproductive motivation. This parameter sets the interpolation array weight that defines how fast the reproduction motivation the_neurobio::state_reproduce is allowed to raise when the age of the agent exceeds the reproductive age. For details see the_neurobio::appraisal_motivation_modulation_non_genetic().
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::sex_male_modulation_reproduce_geno
= reshape ([NO, NO, NO, NO, NO, NO, YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for sex modulation coefficient affecting **reproduction** motivation state in **males**.*
- real(srp), parameter, public commondata::sex_male_modulation_reproduce_gerror_cv = 0.1_SRP

  *Gaussian error parameter (cv) for the sex modulation coefficient affecting **reproduction** motivation state in **males**.*
- logical, dimension(max_nalleles, n_chromosomes), parameter, public commondata::sex_female_modulation_reproduce_ger
= reshape ([NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, YES, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO, NO], [MAX_NALLELES,N_CHROMOSOMES], [NO], [2,1])

  *The genotype structure for sex modulation coefficient affecting **reproduction** motivation state in **females**.*
- real(srp), parameter, public commondata::sex_female_modulation_reproduce_gerror_cv = 0.1_SRP

  *Gaussian error parameter (cv) for the sex modulation coefficient affecting **reproduction** motivation state in **females**.*
- real(srp), parameter, public commondata::attention_switch_hunger_light = 0.0_SRP

  Baseline attention switches $\Psi_{i,j}$ control which perception components (i) can affect each of the motivational state (j). They should be defined for all combinations between the perception components (light, depth... food... predators... etc) and the motivational states. See Cognitive architecture.
- real(srp), parameter, public commondata::attention_switch_hunger_depth = 0.0_SRP

  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_hunger_food_dir = 1.0_SRP

  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_hunger_food_mem = 1.0_SRP

  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_hunger_conspec = 1.0_SRP

  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_hunger_pred_dir = 0.0_SRP

  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_hunger_predator = 0.0_SRP

  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_hunger_stomach = 1.0_SRP

  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_hunger_bodymass = 1.0_SRP

  Baseline attention switch, see commondata::attention_switch_hunger_light.

- real(srp), parameter, public commondata::attention_switch_hunger_energy = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_hunger_age = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_hunger_reprfac = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_avoid_act_light = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_avoid_act_depth = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_avoid_act_food_dir = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_avoid_act_food_mem = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_avoid_act_conspec = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_avoid_act_pred_dir = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_avoid_act_predator = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_avoid_act_stomach = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_avoid_act_bodymass = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_avoid_act_energy = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_avoid_act_age = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_avoid_act_reprfac = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_reproduce_light = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_reproduce_depth = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_reproduce_food_dir = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_reproduce_food_mem = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_reproduce_conspec = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_reproduce_pred_dir = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_reproduce_predator = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_reproduce_stomach = 0.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_reproduce_bodymass = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_reproduce_energy = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_reproduce_age = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), parameter, public commondata::attention_switch_reproduce_reprfac = 1.0_SRP
  Baseline attention switch, see commondata::attention_switch_hunger_light.
- real(srp), dimension(∗), parameter, public commondata::attention_modulation_curve_abscissa =[0.0_↩ SRP, 0.3_SRP, 0.5_SRP, 1.0_SRP]
  *The array defining the **abscissa** (X) of the nonparametric function that defines the **attention modulation curve** by the current Global Organismic State (GOS).*

- real(srp), dimension(∗), parameter, public commondata::attention_modulation_curve_ordinate =[1.0_SRP, 0.98_SRP, 0.9_SRP, 0.0_SRP]

  *The array defining the **ordinate** (Y) of the nonparametric function that defines the **attention modulation curve** by the current Global Organismic State (GOS).*
- real(srp), dimension(∗), parameter, public commondata::motivation_compet_threshold_curve_abscissa = [0.0_SRP, 0.2_SRP, 0.60_SRP, 0.80_SRP, 0.90_SRP, 1.0_SRP, 1.1_SRP]

  *The array defining the **abscissa** (X) of the nonparametric function curve that defines the threshold for motivation competition in GOS.*
- real(srp), dimension(∗), parameter, public commondata::motivation_compet_threshold_curve_ordinate = [1.0_SRP, 0.3_SRP, 0.04_SRP, 0.01_SRP, 0.001_SRP, 0.0_SRP, 0.0_SRP]

  *The array defining the **ordinate** (Y) of the nonparametric function curve that defines the threshold for motivation competition in GOS.*
- real(srp), parameter, public commondata::arousal_gos_dissipation_factor = 0.5_SRP

  *Spontaneous arousal dissipation level when a simple **fixed** dissipation factor pattern is used. At each step, `gos_arousal` is reduced by a constant factor, AROUSAL_GOS_DISSIPATION_FACTOR` (e.g. reduced by 0.5) independently on the current GOS time step.*
- real(srp), dimension(∗), parameter, public commondata::arousal_gos_dissipation_nonpar_abscissa = [1.0, 2.00, 5.00, 10.0, 15.0, 18.0, 20.0]

  *This is the array defining the **abscissa** (X) of the nonparametric spontaneous arousal dissipation factor function involving polynomial (or linear) interpolation is used.*
- real(srp), dimension(∗), parameter, public commondata::arousal_gos_dissipation_nonpar_ordinate = [1.0, 0.98, 0.80, 0.40, 0.22, 0.18, 0.17]

  *This is the array defining the **ordinate** (Y) of the nonparametric spontaneous arousal dissipation factor function involving polynomial (or linear) interpolation is used.*
- real(srp), public commondata::global_rescale_maximum_motivation

  Global maximum sensory information that is updated for the whole population of agents.
- integer, parameter, public commondata::history_size_behaviours = HISTORY_SIZE_SPATIAL

  The size of the behaviour labels history stack, i.e. for how many time steps should the stack remember record the behaviour labels.
- real(srp), parameter, public commondata::probability_reproduction_base_factor = 0.90

  Default weighting factor for the baseline probability of successful reproduction φ. See implementation details for the function the_neurobio::reproduce_do_probability_reproduction_calc().
- real(srp), dimension(∗), parameter, public commondata::probability_reproduction_delta_mass_abscissa = [0.5_SRP, 1.0_SRP, 2.0_SRP]

  *Interpolation grid **abscissa** for the body mass ratio factor that scales the probability of reproduction. For details see the_neurobio::reproduce_do_probability_reproduction_calc() procedure. Commands (template) to produce interpolation plots:*
- real(srp), dimension(∗), parameter, public commondata::probability_reproduction_delta_mass_ordinate = [0.0_SRP, 1.0_SRP, 1.8_SRP]

  *Interpolation grid **ordinate** for the body mass ratio factor that scales the probability of reproduction. For details see the_neurobio::reproduce_do_probability_reproduction_calc() procedure. Commands (template) to produce interpolation plots:*
- real(srp), parameter, public commondata::sex_steroids_reproduction_threshold = 1.3_SRP

  This parameter defines the threshold of the current gonadal steroids level that should exceed the baseline value determined by the genome, for reproduction to be possible.
- real(srp), parameter, public commondata::walk_random_distance_default_factor = 10.0_SRP

  The weighting factor used in calculation of the default random walk distance, in terms of the agent's body length.
- real(srp), parameter, public commondata::walk_random_distance_stochastic_cv = 0.5_SRP

  The coefficient of variation of the distance for stochastic Gaussian random walk (distance is in terms of the agent's body length). Note that for deterministic walk, cv is zero.
- real(srp), parameter, public commondata::walk_random_food_gain_hope = 4.0_SRP

  *The maximum walk distance, **in units of the average distance to food items** in the current perception object, when the expected food gain is calculated on the bases of the current food availability, not using the the_behaviour::hope() function mechanism. If the average walk distance exceeds this value, the expectancy is based on the the_behaviour::hope() function.*
- real(srp), parameter, public commondata::walk_random_food_gain_hope_agentl = 100.0_SRP

  *The maximum walk distance, **in units of the agent body length**, when the expected food gain is calculated on the bases of the current food availability, not using the the_behaviour::hope() function mechanism. If the average walk distance exceeds this value, the expectancy is based on the the_behaviour::hope() function.*
- real(srp), parameter, public commondata::walk_random_pred_risk_hope_agentl = 150.0_SRP

*The maximum walk distance, **in units of the agent body length**, when the expected predation risk is calculated on the basis of the current perception value, not using the the_behaviour::hope() function mechanism. If the average walk distance exceeds this value, the risk expectancy is based on the the_behaviour::hope() function.*

- real(srp), parameter, public commondata::walk_random_vertical_shift_ratio = 0.5_SRP

  The ratio of the vertical to main horizontal shift parameters of the agent's Gaussian random walk. Random walk is done in the "2.5D" mode (the_environment::spatial_moving::rwalk25d()), i.e. with separate parameters for the main horizontal shift and the vertical depth shift. This is done to avoid a potentially too large vertical displacement of the agent during the movement. Thus, the vertical shift distance should normally be smaller than the horizontal shift. The difference between the main horizontal and (the smaller) vertical shifts is defined by this parameter. For example, if it is equal to 0.5, then the vertical depth shift is 0.5 of the main horizontal shift. See the_behaviour::walk_random_do_execute() for more details.
- real(srp), parameter, public commondata::walk_random_vertical_shift_cv_ratio = 1.0_SRP

  *The ratio of the vertical to the main horizontal coefficients of variation for the **vertical** depth distance in the stochastic Gaussian random walk of the agent. Should normally be equal to the main default value set by commondata::walk_random_distance_stochastic_cv. That is 1.0.*
- real(srp), dimension(∗), parameter, public commondata::walk_random_food_hope_abscissa = [0.0_SRP, 1.0_SRP, 3.5_SRP]

  This parameter defines the hope function for calculating the food perception expectancy in the the_behaviour::walk_random behaviour. This is the abscissa for the hope function grid array. Plotting: `htintrpl.exe [0 1 3.5 ] [2, 1, 0]`. See the_behaviour::walk_random_do_this().
- real(srp), dimension(∗), parameter, public commondata::walk_random_food_hope_ordinate = [2.0_SRP, 1.0_SRP, 0.0_SRP]

  This parameter defines the hope function for calculating the food perception expectancy in the the_behaviour::walk_random behaviour. This is the ordinate for the hope function grid array. Plotting: `htintrpl.exe [0 1 3.5 ] [2, 1, 0]`. See the_behaviour::walk_random_do_this().
- real(srp), parameter, public commondata::approach_offset_default = TOLERANCE_HIGH_DEF_SRP

  Default offset for approach, offset is the difference between the approaching agent and the target object.
- real(srp), parameter, public commondata::approach_conspecfic_dilute_general_risk = 0.5_SRP

  Multiplication factor for the general risk of predation used when the agent evaluates the approach to a target conspecific.
- real(srp), parameter, public commondata::approach_conspecfic_dilute_adjust_pair_behind = 0.5_SRP

  Multiplication factor for subjective assessment of the direct risk of predation when the actor agent moves behind the target conspecific, i.e. when the distance between the agent and predator is going to become longer than the distance between the target conspecific and the agent. See the_behaviour::approach_conspecifics_do_this() for details.
- real(srp), dimension(∗), parameter, public commondata::approach_food_gain_compet_factor_abscissa = [0.00_SRP, 0.10_SRP, 1.00_SRP, 1.50_SRP]

  *The grid **abscissa** defining the nonparametric relationship that determines the expected food gain for the "approach conspecifics" behaviour (the_behaviour::approach_conspec class). The function is a weighting factor depending on the ratio of the agent body mass to the target conspecific body mass, for the baseline expected food gain.*
- real(srp), dimension(∗), parameter, public commondata::approach_food_gain_compet_factor_ordinate = [0.00_SRP, 0.01_SRP, 0.50_SRP, 1.00_SRP]

  *The grid **ordinate** defining the nonparametric relationship that determines the expected food gain for the "approach conspecifics" behaviour (the_behaviour::approach_conspec class). The function is a weighting factor depending on the ratio of the agent body mass to the target conspecific body mass, for the baseline expected food gain.*
- real(srp), parameter, public commondata::dist_expect_food_uncertain_fact = 0.7_SRP

  The weighting factor for the distance to the expected food item if the actual distance is uncertain (e.g. no food items currently in perception). See the_behaviour::walk_random_motivations_expect().
- real(srp), parameter, public commondata::history_perception_window_pred = 0.3_SRP

  The size of the memory window that is used in the assessment of predation risk, as a portion of the commondata::history_size_perception. See the_behaviour::walk_random_do_this() and the_behaviour::walk_random_motivations_ex
- real(srp), parameter, public commondata::history_perception_window_food = 0.3_SRP

  The size of the memory window that is used in the assessment of food gain, as a portion of the commondata::history_size_perception. See the_behaviour::walk_random_do_this() and the_behaviour::walk_random_motivations_ex
- real(srp), parameter, public commondata::escape_dart_distance_default_factor = 1.5_SRP

  The weighting factor used in calculation of the default escape distance. The escape distance is equal to the visibility range of the predator multiplied by this factor. Therefore, it should normally exceed 1.0. Otherwise, the escaping object is still within the visibility range of the predator after the escape. See the_behaviour::escape_dart_do_this() for more details.
- real(srp), parameter, public commondata::escape_dart_distance_default_stoch_cv = 0.5_SRP

For stochastic escape, this parameter determines the coefficient of variation of the escape walk. See the_behaviour::escape_dart_do_this() for more details.

- real(srp), parameter, public commondata::up_down_walk_step_stdlength_factor = 4.0_SRP

  The default size of the up and down walks performed by the GO_DOWN_DEPTH and GO_UP_DEPTH, see the_behaviour::go_down_depth and the_behaviour::go_up_depth classes as well as the_behaviour::go_down_do_this() and the_behaviour::go_up_do_this() methods.
- real(srp), parameter, public commondata::migrate_dist_max_step = 800.0_SRP

  The maximum distance (in units of the agent body length) a migrating agent can pass for a single time step of the model. This is basically limited by (an implicit) maximum speed of the agent, in terms of its body length. This parameter sets the limit on the length of a single migration bout.
- real(srp), parameter, public commondata::migrate_random_max_dist_target = 10.0_SRP

  Default maximum distance towards the target environment (in units of the agent's body size) when the agent could emigrate into this target environment. See the_behaviour::behaviour_do_migrate_random() for details.
- real(srp), parameter, public commondata::migrate_dist_penetrate_offset = 1.0_SRP

  The offset, in terms of the body length of the actor agent, for initial penetrating into the target environment when the agent is migrating into this environment. See the_environment::migrate_do_this().
- real(srp), parameter, public commondata::migrate_food_gain_maximum_hope = 2.0_SRP

  This parameter defines the hope function for calculating the food gain expectancy in the migration behaviour. This is the maximum value of the hope function that is achieved at zero ratio of the old to new food gain memory values. Plotting: `htintrpl.exe [0 1 3.5] [2 1 0]`. See the_behaviour::migrate_do_this().
- real(srp), parameter, public commondata::migrate_food_gain_ratio_zero_hope = 3.5_SRP

  This parameter defines the hope function for calculating the food gain expectancy in the migration behaviour. This is the maximum ratio of the old to new food gain memory values that leads to virtually zero value of the hope function. Plotting: `htintrpl.exe [0 1 3.5] [2 1 0]`. See the_behaviour::migrate_do_this().
- real(srp), parameter, public commondata::migrate_predator_maximum_hope = 2.0_SRP

  This parameter defines the hope function for calculating the general predation risk expectancy in the migration behaviour. This is the maximum value of the hope function that is achieved at zero ratio of the old to new predation values in the memory stack. Plotting: `htintrpl.exe [0 1 3.5] [2 1 0]`. See the_behaviour::migrate_do_this().
- real(srp), parameter, public commondata::migrate_predator_zero_hope = 3.5_SRP

  This parameter defines the hope function for calculating the general predation risk expectancy in the migration behaviour. This is the maximum ratio of the old to new predation values in the memory stack that leads to virtually zero value of the hope function. Plotting: `htintrpl.exe [0 1 3.5] [2 1 0]`. See the_behaviour::migrate_do_this().
- real(srp), dimension(∗), parameter, public commondata::behav_walk_step_stdlen_static = [1.0_SRP, 10.0_SRP, 25.0_SRP, 50.0_SRP, 100.0_SRP]

  This parameter array defines the repertoire of predetermined static walk step sizes, in units of the agent's body length, for the the_behaviour::walk_random behavioural unit as executed in the the_behaviour::behaviour::walk_random class level. See the_behaviour::behaviour::select() method for details.
- real(srp), dimension(∗), parameter, public commondata::behav_go_up_down_step_stdlen_static = [10.↩ 0_SRP, 20.0_SRP, 50.0_SRP, 75.0_SRP, 100.0_SRP]

  This parameter array defines the step sizes, in units of the agent's body length, for the the_behaviour::go_down_depth and the_behaviour::go_up_depth behavioural unit as executed in the the_behaviour::behaviour::depth_down and the_behaviour::behaviour::depth_up class level(s). See the_behaviour::behaviour::select() method for details.

Parameters of the Genetic Algorithm

- real(srp), parameter, public commondata::ga_reproduce_pr = 0.05_SRP

  Percentage of the best reproducing agents in the pre-evolution phase.
- integer, parameter, public commondata::ga_reproduce_n = int(POPSIZE ∗ GA_REPRODUCE_PR)

  Upper limit on the number of reproducing individuals in the fixed-fitness pre-evolution phase.
- integer, parameter, public commondata::ga_fitness_dead = 400000000

  Fitness value ascribed to dead agent in pre-evol. See the_individual::individual_agent::fitness_calc(). Also note that huge(integer)) = 2147483647.
- integer, parameter, public commondata::ga_fitness_select = 900

  Fitness threshold for the inclusion of the agent into the reproducing elite group.
- real(srp), parameter, public commondata::ga_reproduce_min_prop = 0.05_SRP

  Minimum proportion of reproducing agents, but note that the number of number reproducers cannot be smaller than the absolute minimum commondata::ga_reproduce_n_min. See the_population::population::ga_reproduce_max().
- integer, parameter, public commondata::ga_reproduce_n_min = 20

  Absolute minimum number of reproducing agents in the adaptive GA procedure. See the_population::population::ga_reproduce_max().

10.3.1 Detailed Description

This module defines common global parameters and objects for the AHA Model. It also contains a general overview of the AHA Model in Doxygen notation.

Author

Sergey Budaev sergey.budaev@uib.no

Jarl Giske jarl.giske@uib.no

Date

2016-2017

10.4 m_env.f90 File Reference

The environmental objects of the AHA Model.

Data Types

- type the_environment::spatial

  Definition of a spatial object. Spatial object determines the position of the agent, food items and other things in the simulated space. Here we use continuous 3D environment (real type coordinates)
- type the_environment::spatial_moving

  Definition of a movable spatial object. It extends the the_environment::spatial object, but also adds its previous position history in stack-like arrays. The history is maintained with the commondata::add_to_history() subroutine, adding a single element on the top (end) of the history stack.
- type the_environment::environment

  *Definition of the overall **environment**. Environment is a general container for all habitats, patches and other similar objects where the whole life of the agents takes place. Environment just provides the limits for all these objects.*
- type the_environment::food_item

  Definition of a single food item. Food item is a spatial object that has specific location in space. It can be "created" and eaten ("disappear"). Food item is an immobile SPATIAL object that has a position in 3D space.
- type the_environment::food_resource

  Definition of the super-type FOOD resource type. This is a superclass, several sub-classes can be defined for different kinds of food and prey objects.
- type the_environment::predator

  *Definition of the PREDATOR objects. **Predator** is a moving agent that hunts on the evolving AHA agents but its internal structure is very simplistic (although we can in principle doit as a full AHA complexity with genome, GOS etc...).*
- type the_environment::habitat

  *Definition of the **environment habitat** HABITAT object. There can potentially be of several types of habitats (patches etc.), so the superclass HABITAT defines the most general properties and procedures. More specific procedures are defined in sub-objects. Such procedures can be overriden from super-object to sub-objects providing for procedure polymorphism.*
- interface the_environment::light_surface

  Calculate surface light intensity (that is subject to diel variation) for specific time step of the model. Irradiance can be stochastic if an optionallogical stochastic flag is set to TRUE.
- interface the_environment::light_depth

  Calculate underwater background irradiance at specific depth.
- interface the_environment::visual_range

  Calculate visual range of predator using Dag Aksnes's procedures srgetr(), easyr() and deriv().
- interface the_environment::visual_range_new

  Calculate visual range of predator using Dag Aksnes's procedures srgetr(), easyr() and deriv().
- interface the_environment::dist

  Internal distance calculation backend engine.

- interface the_environment::join

  An alias for the the_environment::food_resources_collapse_global_object() method for joining food resources into a single global food resource out of the global array the_environment::global_habitats_available. See the_environment::unjoin() for how to unjoin an array of food resources back into an array.
- interface the_environment::unjoin

  An alias to the_environment::food_resources_update_back_global_object() method to transfer (having been modified) food resource objects out from the single united object back to the global array the_environment::global_habitats_available. See the_environment::join() for how to join an array of food resources into a single global object.
- interface the_environment::assemble

  *Interface to the procedure to **assemble** the global array of habitat objects the_environment::global_habitats_available from a list of separate habitat object components. This call.*
- interface the_environment::disassemble

  *Interface to the procedure to **disassemble** the global habitats objects array the_environment::global_habitats_available back into separate habitat object components. See the_environment::global_habitats_disassemble() for the backend implementation.*
- interface the_environment::operator(.cat.)

  Interface operator to concatenate two arrays of the spatial the_environment::spatial or spatial moving the_environment::spatial_moving objects.
- interface the_environment::operator(.catloc.)

  *Interface operator to concatenate the **location** components of two arrays ofthe_environment::spatial **class** objects.*
- interface the_environment::operator(.within.)

  Interface operators `.within.` for testing whether a spatial object (first argument lies within an environment (second argument). Usage:
- interface the_environment::operator(.contains.)

  Interface operators `.contains.` for testing whether an environment object (first argument) contains a `SPATIAL` object (second argument). Usage:
- interface the_environment::operator(.above.)

  Interface operators .above. for spatial objects. Usage:
- interface the_environment::operator(.below.)

  Interface operators .below. for spatial objects. Usage:
- interface the_environment::operator(-)

  Interface operator "-" for the the_environment::environment spatial container objects. Return an environment object that is shrunk by a fixed value in the 2D XxY plane. See `the_environment::environment_shrink_xy_fixed()`. The operator can be used as follows:

Modules

- module the_environment

  Definition of environmental objects.

Functions/Subroutines

- elemental subroutine the_environment::spatial_create_empty (this)

  These are public access functions, but probably we don't need to allow public access to functions inside generic interfaces.
- subroutine the_environment::environment_whole_build_vector (this, min_coord, max_coord)

  Create the highest level container environment. Set the size of the 3D environment container as two coordinate vectors setting the minimum and maximum coordinate limits: `min_coord(1)` for x, `min_coord(2)` for y, `min_coord(3)` for z The size of the environment should be set from parameter vectors in `COMMONDATA`.
- subroutine the_environment::environment_whole_build_object (this, min_coord, max_coord)

  Create the highest level container environment. Set the size of the 3D environment container as two coordinate vectors setting the minimum and maximum coordinate limits. The parameters `min_coord` and `max_coord` are SPATIAL objects.
- subroutine the_environment::environment_build_unlimited (this)

*Build an **unlimited environment**, with the spatial coordinates limited by the maximum machine supported values based on the intrinsic `huge` function.*

- type(environment) function the_environment::environment_shrink_xy_fixed (this, shrink_value)

  Return an environment object that is shrunk by a fixed value in the 2D XxY plane.
- type(spatial) function the_environment::environment_get_minimum_obj (this)

  *Function to get the **minimum** spatial limits (coordinates) of the environment.*
- type(spatial) function the_environment::environment_get_maximum_obj (this)

  *Function to get the **maximum** spatial limits (coordinates) of the environment.*
- elemental real(srp) function the_environment::environment_get_minimum_depth (this)

  *Get the **minimum depth** in this environment.*
- elemental real(srp) function the_environment::environment_get_maximum_depth (this)

  *Get the **maximum depth** in this environment.*
- pure type(spatial) function, dimension(dim_environ_corners) the_environment::environment_get_corners_2dxy (this, ref_depth, offset)

  Get the corners of the environment in the 2D X Y plane. This is a very simplistic procedure that works only with the box environmental objects.
- elemental logical function the_environment::environment_check_located_within_3d (this, check_object)

  Check if a spatial object is actually within this environment.
- type(spatial) function the_environment::environment_random_uniform_spatial_3d (this)

  *Generate a random spatial object with the uniform distribution within (i.e. bound to) **this** environment.*
- type(spatial) function the_environment::environment_random_uniform_spatial_2d (this, fixdepth)

  *Generate a random spatial object with the uniform distribution within (i.e. bound to) **this** environment, the third depth coordinate is fixed.*
- type(spatial) function, dimension(num) the_environment::environment_random_uniform_spatial_vec_3d (this, num)

  *Generate a vector of random spatial objects with the uniform distribution within (i.e. bound to) **this** environment.*
- type(spatial) function, dimension(size(fixdep_array)) the_environment::environment_random_uniform_spatial_vec_2d (this, fixdep_array)

  *Generate a vector of random spatial objects with the uniform distribution within (i.e. bound to) **this** environment. The third, depth coordinate is non-stochastic, and provided as an array parameter.*
- type(spatial) function, dimension(num) the_environment::environment_random_gaussian_spatial_3d (this, num, centroid, variance)

  *Generates a vector of random spatial object with Gaussian coordinates within (i.e. bound to) **this** environment.*
- type(spatial) function, dimension(num) the_environment::environment_random_gaussian_spatial_2d (this, num, centroid, fixdepth, variance, variance_depth)

  *Generates a vector of random spatial object with Gaussian coordinates within (i.e. bound to) **this** environment. The depth coordinate is set separately and can be non-random (fixed for the whole output array) or Gaussian with separate variance.*
- type(spatial) function centroid_urandom (fixed_depth)

  *Make a random centroid with fixed depth bound within **this** environment.*
- subroutine the_environment::environment_get_nearest_point_in_outside_obj (this, outside_object, offset_↩ into, point_spatial, point_dist)

  Get the spatial point position within this environment that is nearest to an arbitrary spatial object located outside of the this environment. If the spatial object is actually located in this environment, return its own position.
- subroutine the_environment::spatial_fix_position_3d_s (this, x, y, depth)

  Place spatial object into a 3D space, define the object's current coordinates.
- elemental subroutine the_environment::spatial_fix_position_3d_o (this, location)

  Place spatial object into a 3D space, define the object's current coordinates.
- elemental subroutine the_environment::spatial_make_missing (this)

  Assign all commondata::missing` coordinates to the_environment::spatial object.
- elemental type(spatial) function the_environment::spatial_get_current_pos_3d_o (this)

  Get the current spatial position of a `SPATIAL` object.

- pure real(srp) function, dimension(dimensionality_default) the_environment::spatial_get_current_pos_3d_v (this, vector)

  Get the current spatial position of a `SPATIAL` object.
- elemental real(srp) function the_environment::spatial_get_current_pos_x_3d (this)

  Get the current `X` position of a `SPATIAL` object.
- elemental real(srp) function the_environment::spatial_get_current_pos_y_3d (this)

  Get the current `Y` position of a `SPATIAL` object.
- elemental real(srp) function the_environment::spatial_get_current_pos_d_3d (this)

  Get the current `DEPTH` position of a `SPATIAL` object.
- real(srp) function the_environment::spatial_calc_irradiance_at_depth (this, time_step_model)

  Calculate the illumination (background irradiance) at the depth of the spatial object at an arbitrary time step of the model.
- real(srp) function the_environment::spatial_visibility_visual_range_cm (this, object_area, contrast, time_↩ step_model)

  Calculate the visibility range of a spatial object. Wrapper to the the_environment::visual_range() function. This function calculates the distance from which this object can be seen by a visual object (e.g. predator or prey).
- pure integer function the_environment::spatial_get_environment_in_pos (this, environments_array)

  Identify in which environment from the input list this spatial agent is currently in. Example call:
- subroutine the_environment::spatial_moving_fix_position_3d_v (this, x, y, depth)

  Place spatial movable object into a 3D space, define the object's current coordinates, but first save previous coordinates.
- elemental subroutine the_environment::spatial_moving_fix_position_3d_o (this, location)

  Place spatial movable object into a 3D space, define the object's current coordinates, but first save previous coordinates.
- elemental subroutine the_environment::spatial_moving_repeat_position_history_3d (this)

  Repeat (re-save) the current position into the positional history stack.
- elemental real(srp) function the_environment::spatial_distance_3d (this, other)

  Calculate the Euclidean distance between two spatial objects. This is a type-bound function.
- pure type(spatial) function, dimension(:), allocatable the_environment::spatial_stack2arrays (a, b)

  Concatenate two arrays of the_environment::spatial objects `a` and `b`. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.
- pure type(spatial_moving) function, dimension(:), allocatable the_environment::spatial_moving_stack2arrays (a, b)

  Concatenate two arrays of the_environment::spatial_moving objects `a` and `b`. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.
- pure type(spatial) function, dimension(:), allocatable the_environment::spatial_class_stack2arrays_locs (a, b)

  *Concatenate the **location** components of two arrays of the_environment::spatial class objects a and b. This procedure uses array slices which would be faster in most cases than the intrinsic `[a,b]` method.*
- elemental real(srp) function the_environment::dist3d (this, other)

  *This is a **non-type-bound** version of the distance calculation function.*
- elemental real(srp) function the_environment::spatial_self_distance_3d (this, from_history)

  Calculate the Euclidean distance between the current and previous position of a single spatial object.
- elemental real(srp) function the_environment::spatial_moving_self_distance_3d (this, from_history)

  *Calculate the Euclidean distance between the current and previous position of a single spatial movable object. Optionally, it also calculates the total distance traversed during the `from_history` points from the history stack along with the distance from the current position and the last historical value. For example, to calculate the **average** distance throughout the whole history (`HISTORY_SIZE_SPATIAL` points) do this:*
- elemental subroutine the_environment::spatial_moving_create_3d (this)

  Create a new spatial moving object. Initially it has no position, all coordinate values are `MISSING` or `INVALID` for real type coordinates.
- elemental subroutine the_environment::spatial_moving_clean_history_3d (this)

  Create a new empty history of positions for spatial moving object. Assign all values to the MISSING value code.
- elemental subroutine the_environment::spatial_moving_go_up (this, step)

*The spatial moving object **ascends**, goes up the depth with specific fixed step size.*

- elemental subroutine the_environment::spatial_moving_go_down (this, step)

  *The spatial moving object **decends**, goes down the depth with specific fixed step size.*

- subroutine the_environment::spatial_moving_randomwalk_gaussian_step_3d (this, meanshift, cv_shift, environment_limits)

  Implements an optionally environment-restricted Gaussian random walk in 3D.

- subroutine the_environment::spatial_moving_randomwalk_gaussian_step_25d (this, meanshift_xy, cv_↩ shift_xy, meanshift_depth, cv_shift_depth, environment_limits)

  Implements an optionally environment-restricted Gaussian random walk in a "2.5 dimensions", i.e. 2D x y with separate walk parameters for the third depth dimension.

- subroutine the_environment::spatial_moving_corwalk_gaussian_step_3d (this, target, meanshift, cv_shift, is_away, ci_lim, environment_limits, is_converged, debug_reps)

  *Implements an optionally environment-restricted **correlated directional** Gaussian random walk in 3D towards (or away of) an the_environment::spatial class `target` object.*

- subroutine the_environment::spatial_moving_corwalk_gaussian_step_25d (this, target, meanshift_xy, cv_↩ shift_xy, meanshift_depth, cv_shift_depth, is_away, ci_lim, environment_limits, is_converged, debug_reps)

  *Implements an optionally environment-restricted **correlated directional** Gaussian random walk in 3D towards (or away of) an the_environment::spatial class `target` object.*

- subroutine the_environment::spatial_moving_dirwalk_gaussian_step_3d (this, target, meanshift, cv_shift, environment_limits)

  *Implements an optionally environment-restricted **directional** Gaussian random walk in 3D towards a `target` the_environment::spatial class object.*

- real(srp) function updated_position (coord_target, coord_object)

  Calculate a Gaussian random updated coordinate for multidimensional Gaussian targeted random walk along any of the dimensions. The dalta shift has value and variance but may be either forward or backward so as to minimise the distance from the target.

- subroutine the_environment::spatial_moving_dirwalk_gaussian_step_25d (this, target, meanshift_xy, cv_↩ shift_xy, meanshift_depth, cv_shift_depth, environment_limits)

  *Implements an optionally environment-restricted **directional** Gaussian random walk in "2.5"-D towards a `target` the_environment::spatial class object. i.e. 2D x y with separate walk parameters for the third depth dimension.*

- real(srp) function updated_position (coord_target, coord_object, meanshift, cv_shift)

  Calculate a Gaussian random updated coordinate for multidimensional Gaussian targeted random walk along any of the dimensions. The dalta shift has value and variance but may be either forward or backward so as to minimise the distance from the target.

- subroutine the_environment::rwalk3d_array (this, dist_array, cv_array, dist_all, cv_all, environment_limits, n↩ _walks)

  Perform one or several steps of random walk by an array of the_environment::spatial_moving class objects. This is a 3D version with the same walk parameters for the horizontal XxY plane and depth.

- subroutine the_environment::rwalk25d_array (this, dist_array_xy, cv_array_xy, dist_array_depth, cv_array_↩ depth, dist_all_xy, cv_all_xy, dist_all_depth, cv_all_depth, environment_limits, n_walks)

  Perform one or several steps of random walk by an array of the_environment::spatial_moving class objects. This is a 2.5D version with separate walk parameters for the horizontal XxY plane and depth.

- elemental logical function the_environment::spatial_check_located_within_3d (this, environment_limits)

  Function to check if this spatial object is located within an area set by an environmental object (parameter). This should be similar to an analogous function defined for the environment object.

- elemental logical function the_environment::spatial_check_located_below (this, check_object)

  *Logical function to check if the **argument** spatial object(s) (`check_object`) is (are) located **below** the **this** reference spatial object. Elemental function that also works with arrays. Use as:*

- elemental logical function the_environment::spatial_check_located_above (this, check_object)

  *Logical function to check if the **argument** spatial object(s) (`check_object`) is (are) located **above** the **this** reference spatial object. Elemental function that also works with arrays. Use as:*

- type(spatial) function the_environment::spatial_get_nearest_object (this, neighbours, number)

  *Determine the nearest spatial object to **this** spatial object among an array of other spatial objects.*

- integer function the_environment::spatial_get_nearest_id (this, neighbours, object)

  *Determine the nearest spatial object to **this** spatial object among an array of other spatial objects.*

- subroutine the_environment::habitat_make_init (this, coord_min, coord_max, label, otherrisks, eggmortality, predators_number, loc_predators, food_abundance, loc_food, sizes_food)

  Make an instance of the habitat object (an environment superset).
- character(len=label_length) function the_environment::habitat_name_get (this)

  Return the name of the habitat.
- real(srp) function the_environment::habitat_get_risk_mortality (this)

  Get the mortality risk associated with this habitat.
- real(srp) function the_environment::habitat_get_risk_mortality_egg (this)

  Get the egg mortality risk associated with this habitat.
- subroutine the_environment::habitat_save_predators_csv (this, csv_file_name, is_success)

  Save the predators with their characteristics into a CSV file.
- subroutine the_environment::save_dynamics (maxdepth, csv_file_name, is_success)

  Save diagnostics data that shows the dynamics of the light and the average depth of the food items, light at the average depth of the food items etc at each time step of the model.
- type(spatial) function the_environment::environment_centre_coordinates_3d (this, nodepth)

  Determine the centroid of the environment.
- real(srp) function, private the_environment::visual_range_scalar (irradiance, prey_area, prey_contrast)

  Wrapper for calculating visual range of a fish predator using the Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`. See `srgetr()` for computational details.
- real(srp) function, dimension(size(prey_area)), private the_environment::visual_range_vector (irradiance, prey_area, prey_contrast_vect, prey_contrast)

  Wrapper for calculating visual range of a fish predator using the Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`. See `srgetr()` for computational details.
- elemental real(srp) function the_environment::visual_range_fast (irradiance, prey_area, prey_contrast)

  *Wrapper for calculating visual range of a fish predator using the Dag Aksnes's procedures `srgetr()`, `easyr()` and `deriv()`. This is a new **elemental** and parallel-ready visual range function wrapper making use the elemental-procedures based computational backend. See notes on `visual_range_scalar()` and `srgetr()` for computational details.*
- elemental real(srp) function, private the_environment::light_surface_deterministic (tstep)

  Calculate deterministic surface light at specific time step of the model. Light (`surlig`) is calculated from a sine function. Light intensity just beneath the surface is modelled by assuming a 50 % loss by scattering at the surface:
- real(srp) function, private the_environment::light_surface_stochastic_scalar (tstep, is_stochastic)

  Calculate stochastic surface light at specific time step of the model. Light (`surlig`) is calculated from a sine function. Light intensity just beneath the surface is modelled by assuming a 50 % loss by scattering at the surface:
- real(srp) function, dimension(size(tstep)), private the_environment::light_surface_stochastic_vector (tstep, is_stochastic)

  Calculate stochastic surface light at specific time step of the model.
- real(srp) function, private the_environment::light_depth_integer (depth, surface_light, is_stochastic)

  Calculate underwater light at specific depth given specific surface light.
- real(srp) function, private the_environment::light_depth_real (depth, surface_light, is_stochastic)

  Calculate underwater light at specific depth given specific surface light.
- elemental real(srp) function the_environment::dist_scalar (x1, x2, y1, y2, z1, z2)

  *Calculate distance between 3D or 2D points. This is a function engine for use within type bound procedures. **Example** (dist_scalar):*
- pure real(srp) function the_environment::dist_vector_nd (cvector1, cvector2)

  Calculate distance between N-dimensional points. This is a function engine for use within other type bound procedures.
- pure real(srp) function the_environment::dist2_vector (cvector1, cvector2)

  Calculate the squared distance between two N-dimensional points.
- pure real(srp) function the_environment::vect_magnitude (vector)

  Calculate the magnitude of an arbitrary N-dimensional vector. This is a raw vector backend.
- elemental real(srp) function the_environment::dist2step (average_distance, dimensionality)

Calculate the unit step along a single coordinate axis given the average distance between any two points in a N-dimensional Gaussian random walk.

- elemental subroutine the_environment::food_item_create (this)

  Create a single food item at an undefined position with default size.
- elemental subroutine the_environment::food_item_make (this, location, size, iid)

  Make a single food item, i.e. place it into a specific position in the model environment space and set the size.
- logical function the_environment::food_item_capture_success_stochast (this, prob)

  *Stochastic outcome of **this** food item capture by an agent. Returns TRUE if the food item is captured.*
- real(srp) function the_environment::food_item_capture_probability_calc (this, distance, time_step_model)

  *Calculate the probability of capture of **this** food item by a predator agent depending on the distance between the agent and this food item.*
- real(srp) function the_environment::food_item_visibility_visual_range (this, object_area, contrast, time_↩ step_model)

  Calculate the visibility range of this food item. Wrapper to the the_environment::visual_range() function. This function calculates the distance from which this food item can be seen by a predator (i.e. the default predator's visual range).
- real(srp) function the_environment::minimum_depth_visibility (target_range, depth_range_min, depth_↩ range_max, object_area, object_contrast, time_step_model)

  Find the depth at which the visibility of a spatial object becomes smaller than a specific distance value `target_↩ range`.
- real(srp) function visibility_loc (depth)

  This function calculates the visibility range of the spatial object at the depth given by the argument `depth`. This function is internal to the_environment::minimum_depth_visibility().
- real(srp) function visibility_loc_diff (depth)

  This is a wrapper function that calculates the visibility range minus target minimum distance. This function is internal to the_environment::minimum_depth_visibility().
- elemental subroutine the_environment::food_item_disappear (this)

  Make the food item "disappear" and take the "eaten" state, i.e. impossible for consumption by the agents.
- elemental logical function the_environment::food_item_is_eaten_unavailable (this)

  Logical check-indicator function for the food item being eaten and not available.
- elemental logical function the_environment::food_item_is_available (this)

  Logical check-indicator function for the food item being available.
- elemental real(srp) function the_environment::food_item_get_size (this)

  Get the size component of the food item object.
- elemental real(srp) function the_environment::size2mass_food (radius)

  Calculate the mass of a food item, the non-OO backend.
- elemental real(srp) function the_environment::mass2size_food (mass)

  Calculate the size (radius) of a food item, a reverse function of the_environment::size2mass_food():
- elemental real(srp) function the_environment::food_item_get_mass (this)

  Calculate and get the mass of the food item.
- elemental subroutine the_environment::food_item_set_iid (this, iid)

  Set unique id for the food item object.
- elemental subroutine the_environment::food_item_clone_assign (this, the_other)

  Clone the properties of this food item to another food item.
- elemental integer function the_environment::food_item_get_iid (this)

  Get the unique id of the food item object.
- pure subroutine the_environment::food_resource_make (this, label, abundance, locations, sizes)

  Make food resource object. This class standard constructor.
- elemental integer function the_environment::food_resource_get_abundance (this)

  Get the number of food items in the food resource.
- elemental character(len=label_length) function the_environment::food_resource_get_label (this)

  Get the label of the this food resource.
- pure subroutine the_environment::food_resource_destroy_deallocate (this)

Delete and deallocate food resource object. This class standard destructor.

- pure type(spatial) function, dimension(size(this%food)) the_environment::food_resource_locate_3d (this)

  Get the location object array (array of `SPATIAL` objects) of a food resource object.

- real(srp) function the_environment::food_resource_calc_average_distance_items (this, n_sample)

  Calculate the average distance between food items within a resource. e.g. to compare it with the agent's random walk step size.

- subroutine the_environment::food_resource_replenish_food_items_all (this, replace)

  Replenish and restore food resource. The food resource is replenished by substituting randomly selected `replace` food items or all items if `replace` is omitted or exceeds the actual number of food items. Unlike the the_environment::food_resource::make() method, the sizes and the positions of the food items within the resource are reused from the previous positions (previously explicitly set set by the the_environment::food_resource::make() method).

- subroutine the_environment::food_resource_migrate_move_items (this, max_depth, time_step_model)

  This subroutine implements the migration of all the food items in the resource according to the plankton migration pattern from the G1 model (HED18). Briefly, the movement of each of the food items has two components:

- subroutine the_environment::food_resource_rwalk_items_default (this)

  Perform a random walk step for all food items within the food resource. The walk is performed with the default parameters:

- subroutine the_environment::migrate_food_vertical (habitats, time_step_model)

  Migrate food items in a whole array of food resources. The array is normally the the_environment::global_habitats_available.

- subroutine the_environment::rwalk_food_step (habitats)

  Perform a random walk of food items in a whole array of food resources. The array is normally the the_environment::global_habitats_available. This procedure is a wrapper for the_environment::food_resource::rwalk() to do a walk on a whole array of habitats and linked food resources.

- real(srp) function the_environment::center_depth_sinusoidal (tstep, depth)

  This function calculates the target depth for the sinusoidal vertical migration pattern of the food items at each time step of the model. See the_environment::food_resource_migrate_move_items() for the calling procedure.

- subroutine the_environment::food_resource_save_foods_csv (this, csv_file_name, is_success)

  Save characteristics of food items in the resource into a CSV file.

- elemental subroutine the_environment::food_resource_sort_by_size (this, reindex)

  Sort the food resource objects within the array by their sizes. The two subroutines below are a variant of the recursive quick-sort algorithm adapted for sorting real components of the the `FOOD_RESOURCE` object.

- recursive pure subroutine qsort (A)

  `qsort` and `qs_partition_` are the two parts of the recursive sort algorithm `qsort` is the recursive frontend. Sorts an array of food items by size within the resource object.

- pure subroutine qs_partition_size (A, marker)

  `qsort` and `qs_partition_` are the two parts of the recursive sort algorithm `qs_partition_size` is a pivot backend, here it sorts food items within the food resource object by real size components.

- pure subroutine the_environment::food_resource_reset_iid_all (this, start_iid)

  Reset individual iid for the food resource. Individual iids must normally coincide with the array order index. If it is changed after sorting, iids no longer reflect the correct index. So this subroutine resets iids to be coinciding with each food item index.

- subroutine the_environment::reindex_food_resources (resource_1, resource_2, resource_3, resource_↩ 4, resource_5, resource_6, resource_7, resource_8, resource_9, resource_10, resource_11, resource_12, resource_13, resource_14, resource_15, resource_16, resource_17, resource_18, resource_19, resource↩ _20)

  Reset and reindex iids for an input list of several food resources. As the result of this subroutine all food items across all the resources within the whole list will have unique iids.

- subroutine the_environment::food_resources_collapse (food_resource_collapsed, resource_1, resource_↩ 2, resource_3, resource_4, resource_5, resource_6, resource_7, resource_8, resource_9, resource_10, resource_11, resource_12, resource_13, resource_14, resource_15, resource_16, resource_17, resource↩ _18, resource_19, resource_20, reindex, label)

  Collapse several food resources into one. The collapsed resource can then go into the perception system. The properties of the component resources are retained in the collapsed resource.

- type(food_resource) function the_environment::food_resources_collapse_global_object (reindex, label)

Join food resources into a single global food resource out of the global array the_environment::global_habitats_available. See the_environment::unjoin() for how to unjoin an array of food resources back into an array. The joined resource can then go into the perception system. The properties of the component resources are retained in the collapsed resource.

- subroutine the_environment::food_resources_update_back (food_resource_collapsed, resource_1, resource_2, resource_3, resource_4, resource_5, resource_6, resource_7, resource_8, resource_9, resource_10, resource_11, resource_12, resource_13, resource_14, resource_15, resource_16, resource_17, resource_18, resource_19, resource_20, reindex)

  Transfer back the resulting food resources into their original objects out from a collapsed object from `food_resources_collapse`.
- subroutine the_environment::food_resources_update_back_global_object (food_resource_collapsed, reindex)

  Transfer the (having been modified) food resource objects from the single united object `food_resource_collapsed` back to the global array the_environment::global_habitats_available array. See the_environment::join() for how to join an array of food resources into a single global object.
- subroutine the_environment::global_habitats_assemble (habitat_1, habitat_2, habitat_3, habitat_4, habitat_5, habitat_6, habitat_7, habitat_8, habitat_9, habitat_10, habitat_11, habitat_12, habitat_13, habitat_14, habitat_15, habitat_16, habitat_17, habitat_18, habitat_19, habitat_20, reindex)

  Assemble the global habitats objects array the_environment::global_habitats_available from a list of separate habitat objects. This call.
- subroutine the_environment::global_habitats_disassemble (habitat_1, habitat_2, habitat_3, habitat_4, habitat_5, habitat_6, habitat_7, habitat_8, habitat_9, habitat_10, habitat_11, habitat_12, habitat_13, habitat_14, habitat_15, habitat_16, habitat_17, habitat_18, habitat_19, habitat_20, reindex)

  Disassemble the global habitats objects array the_environment::global_habitats_available into separate habitat object.
- subroutine the_environment::spatial_neighbours_distances (this, neighbours, dist, index_vector, ranks, rank_max, error_flag)

  *Calculate the distances between **this** spatial object and an array of its neighbours. Optionally output the distances, sorting index vector and rankings vector for each of these neighbours. Optionally do only partial indexing, up to the order `rank_max` for computational speed. Procedure `ARRAY_INDEX()` from HEDTOOLS is used as the computational backend for partial segmented indexing.*
- elemental subroutine the_environment::predator_make_init (this, body_size, attack_rate, position, label)

  Initialise a predator object.
- subroutine the_environment::predator_label_set (this, label)

  Set label for the predator, if not provided, set it random.
- elemental real(srp) function the_environment::predator_get_body_size (this)

  Accessor function for the predator body size (length).
- elemental real(srp) function the_environment::predator_get_attack_rate (this)

  Accessor function for the predator attack rate.
- real(srp) function the_environment::predator_capture_risk_calculate_fish (this, prey_spatial, prey_length, prey_distance, is_freezing, time_step_model, debug_plot_file)

  Calculates the risk of capture of the fish with the spatial location defined by `prey_spatial` and the body length equal to `prey_length`. This is a backend function bound to the predator rather than prey object.
- subroutine the_environment::predator_capture_risk_calculate_fish_group (this, prey_spatial, prey_length, is_freezing, time_step_model, risk, risk_indexed, index_dist)

  Calculates the risk of capture by a specific predator of an array of the fish agents with the spatial locations array defined by `prey_spatial` and the body length array `prey_length`. This subroutine takes account of both the predator dilution and confusion effects and risk adjusted by the distance towards the predator.
- subroutine adjust_risk_nonpar_noadjust ()

  Adjust the predation risk for confusion and dilution effects.
- subroutine adjust_risk_nonpar_fixed ()

  Adjust the predation risk for confusion and dilution effects.
- subroutine adjust_risk_dilute_nofirst ()

  Adjust the predation risk of a group of N prey agents for predator dilution effect.
- subroutine adjust_risk_dilute_all ()

Adjust the predation risk of a group of N prey agents for predator dilution effect.

- real(srp) function the_environment::predator_visibility_visual_range (this, object_area, contrast, time_step↩ _model)

  Calculate the visibility range of this predator. Wrapper to the the_environment::visual_range() function. This function calculates the distance from which this predator can be seen by a visual object (e.g. the agent).

- real(srp) function the_environment::distance_average (spatial_objects, sample_size)

  Calculates the average nearest neighbour distance amongst an array of spatial objects (class) by sampling `sample_size` of them. The sample size `sample_size` is optional, if not provided set to `SAMPLE_SIZE↩ _DEFAULT=25`.

Visual range calculation backend.

The subroutines the_environment::srgetr(), the_environment::easyr() and the_environment::deriv() should be better isolated into a separate module or form a submodule, but it is not used here as submodules are a F2008 feature not supported by all compiler systems. Anyway, submodule is not essential here.

Note

*Note that all these backend procedures are now **pure** and therefore parallel-safe.*

- elemental subroutine, private the_environment::srgetr (r, c, C0, Ap, Vc, Ke, Eb, IER)

  Obtain visual range by solving the non-linear equation by means of Newton-Raphson iteration and derivation in subroutine the_environment::deriv(). Initial value is calculated in the_environment::easyr(). The calculation is based on the model described in Aksnes & Utne (1997) Sarsia 83:137-147.

- elemental subroutine, private the_environment::easyr (r, C0, Ap, Vc, Ke, Eb)

  Obtain a first estimate of visual range by using a simplified expression of visual range. See `srgetr()` for more details.

- elemental subroutine, private the_environment::deriv (r, F1, FDER, c, C0, Ap, Vc, Ke, Eb)

  Derivation of equation for visual range of a predator. See the_environment::srgetr() for more details.

Computational geometry backend: polygon2D

Rudimentary computational geometry (geo_) procedures, based on 2D polygons (poly2d) with fixed depth or depth ignored.

Note

Manually constructed using 2D X x Y vectors ignoring the depth dimension.

- subroutine the_environment::geo_poly2d_dist_point_to_section (point, sectp1, sectp2, min_dist, point_↩ segment)

  Calculates the minimum distance from a the_environment::spatial class object to a line segment delimited by two the_environment::spatial class endpoints in the 2D XY plane (the depth coordinate is ignored). (The algorithm is partially based on `this`.)

- subroutine the_environment::geo_poly3d_dist_point_to_section (point, sectp1, sectp2, min_dist, point_↩ segment)

  Calculates the minimum distance from a the_environment::spatial class object to a line segment delimited by two the_environment::spatial class endpoints in the 3D XY space. (The algorithm is partially based on `this`.)

- type(spatial) function the_environment::offset_dist (obj_a, obj_b, offset)

  Calculate a the_environment::spatial target with an offset.

Variables

- character(len= ∗), parameter, private the_environment::modname = "(THE_ENVIRONMENT)"
- integer, parameter, private the_environment::dimensionality_default = 3

  Default dimensionality of the environment universe.

- integer, parameter the_environment::dim_environ_corners = 4

  The number of corners for an environment object in the 2D X∗x∗Y plane.

- type(habitat), dimension(:), allocatable, public the_environment::global_habitats_available

  A list (array) of all the the_environment::habitat objects available to the agents. This single array should encompass all the locations that the agent can potentially be in (e.g. migrate from one to another).

10.4.1 Detailed Description

The environmental objects of the AHA Model.

Author

Sergey Budaev sergey.budaev@uib.no
Jarl Giske jarl.giske@uib.no

Date

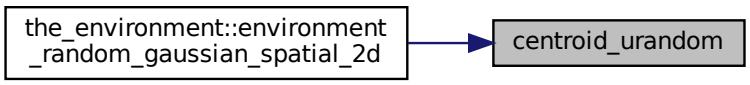
2016-2017

10.4.2 Function/Subroutine Documentation

10.4.2.1 centroid_urandom()

```
type(spatial) function environment_random_gaussian_spatial_2d::centroid_urandom (
            real(srp), optional fixed_depth )
```

Make a random centroid with fixed depth bound within **this** environment.
Definition at line 1596 of file m_env.f90.
Here is the caller graph for this function:



10.4.2.2 updated_position() [1/2]

```
real(srp) function spatial_moving_dirwalk_gaussian_step_3d::updated_position (
            real(srp), intent(in) coord_target,
            real(srp), intent(in) coord_object )
```

Calculate a Gaussian random updated coordinate for multidimensional Gaussian *targeted* random walk along any of the dimensions. The dalta shift has value and variance but may be either forward or backward so as to *minimise the distance from the target*.

Returns

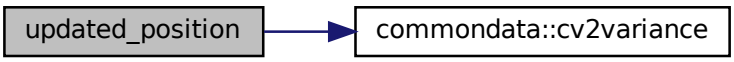
Updated coordinate.

Parameters

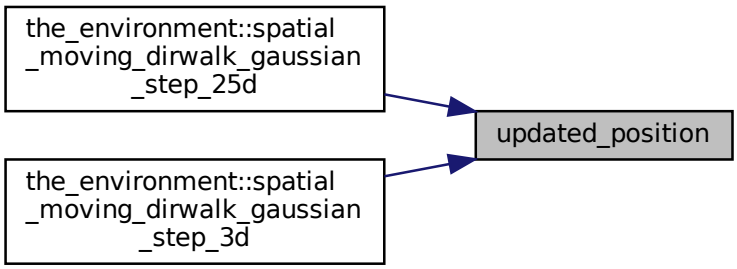
in	*coord_target*	coord_target axis-bound coordinate of the target.
	coord_object	actual axis-bound coordinate of the moving spatial object.
in	*coord_object*	coord_target axis-bound coordinate of the target.
	coord_object	actual axis-bound coordinate of the moving spatial object.

Definition at line 3349 of file m_env.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



10.4.2.3 updated_position() [2/2]

```
real(srp) function spatial_moving_dirwalk_gaussian_step_25d::updated_position (
             real(srp) coord_target,
             real(srp) coord_object,
             real(srp) meanshift,
             real(srp) cv_shift )
```

Calculate a Gaussian random updated coordinate for multidimensional Gaussian *targeted* random walk along any of the dimensions. The dalta shift has value and variance but may be either forward or backward so as to *minimise the distance from the target*.

Returns

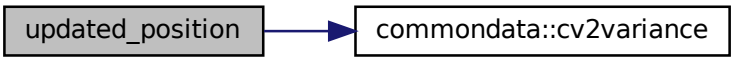
Updated coordinate.

Parameters

coord_target	coord_target axis-bound coordinate of the target.
coord_object	actual axis-bound coordinate of the moving spatial object.
coord_object	coord_target axis-bound coordinate of the target.
coord_object	actual axis-bound coordinate of the moving spatial object.

10.4.2.3.1 Implementation details We first check if the axis-bound distance between the object and the target is less then the mean shift...

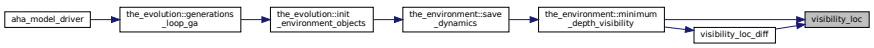
If so, we have now reached the target successfully.
If not, first calculate a Gaussian random shift along this coordinate, the absolute value.
Then we should make sure the object is actually approaching the target. So we choose the direction that minimises the new coordinate-bound distance between the object and the target.
Definition at line 3479 of file m_env.f90.
Here is the call graph for this function:



10.4.2.4 visibility_loc()

```
real(srp) function minimum_depth_visibility::visibility_loc (
            real(srp), intent(in) depth )
```

This function calculates the visibility range of the spatial object at the depth given by the argument `depth`. This function is internal to the_environment::minimum_depth_visibility().
Calculate ambient illumination / irradiance at the depth of this food item at the given time step.
Return the visibility range for this spatial object.
Definition at line 6216 of file m_env.f90.
Here is the caller graph for this function:



10.4.2.5 visibility_loc_diff()

```
real(srp) function minimum_depth_visibility::visibility_loc_diff (
            real(srp), intent(in) depth )
```

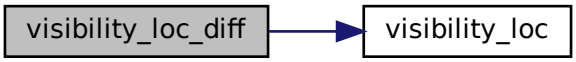
This is a wrapper function that calculates the visibility range minus target minimum distance. This function is internal to the_environment::minimum_depth_visibility().

Parameters

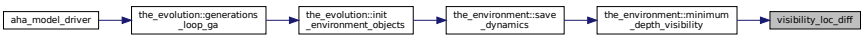
in	*depth*	depth the depth of the spatial object.

Definition at line 6242 of file m_env.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



10.4.2.6 qsort()

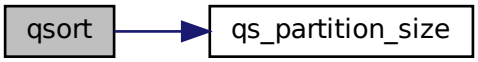
```
recursive pure subroutine food_resource_sort_by_size::qsort (
            type(food_item), dimension(:), intent(inout) A )
```

`qsort` and `qs_partition_` are the two parts of the recursive sort algorithm `qsort` is the recursive frontend. Sorts an array of food items by size within the resource object.

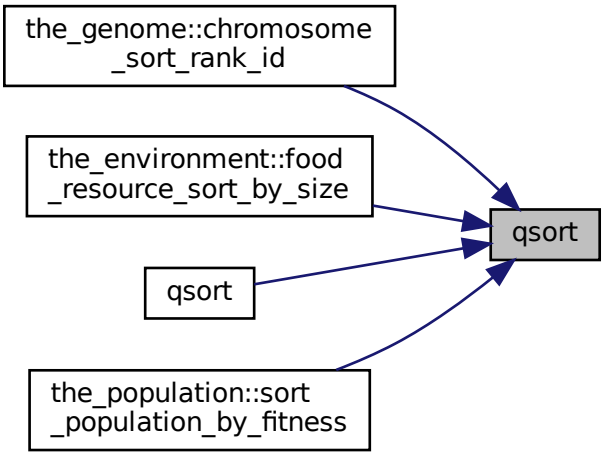
Parameters

<tt>A</tt>	has the same type as the individual component objects of the array-object that we are going to sort.

Definition at line 7013 of file m_env.f90.
Here is the call graph for this function:



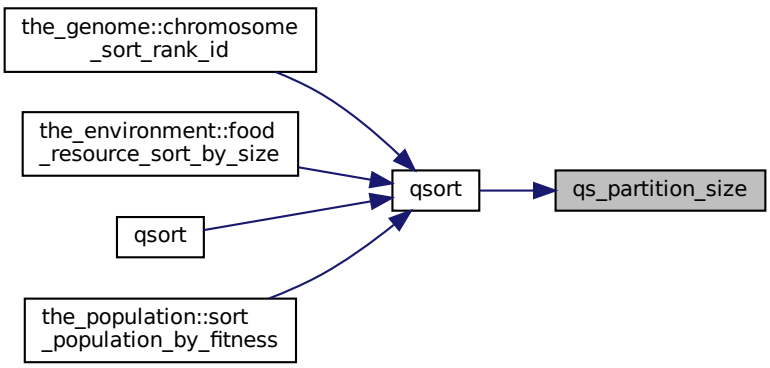
Here is the caller graph for this function:



10.4.2.7 qs_partition_size()

```
pure subroutine food_resource_sort_by_size::qs_partition_size (
            type(food_item), dimension(:), intent(inout) A,
            integer, intent(out) marker )
```

`qsort` and `qs_partition_` are the two parts of the recursive sort algorithm `qs_partition_size` is a pivot backend, here it sorts food items within the food resource object by real size components.
Definition at line 7032 of file m_env.f90.
Here is the caller graph for this function:



10.4.2.8 adjust_risk_nonpar_noadjust()

`subroutine predator_capture_risk_calculate_fish_group::adjust_risk_nonpar_noadjust`
Adjust the predation risk for confusion and dilution effects.
In this version, the adjusted risk is equal to the baseline risk, i.e. there are no specific predator confusion or dilution effects.
See the main container procedure `the_environment::predator_capture_risk_calculate_fish_group()`.

10.4.2.8.1 Implementation details The adjusted risk of predation in this version is simplistic and just equals the baseline risk. So no adjustment is actually made.
Note that the adjusted risk is calculated only for a small subarray within the potentially huge input array of spatial prey agents, commondata::predator_risk_group_select_index_partial maximum elements. All other values are nulls.
Definition at line 10170 of file m_env.f90.

10.4.2.9 adjust_risk_nonpar_fixed()

`subroutine predator_capture_risk_calculate_fish_group::adjust_risk_nonpar_fixed`
Adjust the predation risk for confusion and dilution effects.
This version is based on a fixed pattern linking the rank of the prey agent within the predator's visual field and the adjusted risk. However, the basic pattern scales independently on the specific number of prey agents that the predator sees.
See the main container procedure `the_environment::predator_capture_risk_calculate_fish_group()`.
PROCNAME is the procedure name for logging and debugging

10.4.2.9.1 Notable variables

- **predator_risk_group_dilution_abscissa** is the abscissa for the non-parametric function that links the baseline unadjusted predation risk for any prey agent within a group and the risk estimate that takes account of predator confusion and predator dilution effects. The ordinate of the grid is defined by the parameter array commondata::predator_risk_group_dilution_ordinate.

10.4.2.9.2 Implementation details The grid abscissa for the nonparametric function (see below) depends on the number visible prey agents within the visual field of the predator *N*. It is constructed as a 3-element array (the second point is a middle interval): [1.0, 1 + (*N-1*/2.0), *N*]. The ordinate of the grid is defined by the parameter array commondata::predator_risk_group_dilution_ordinate.

- The estimate of the adjusted predation risk (R_a) for each agent in proximity to this predator then depends on the rank order (r) of the agent in within the visual field of the predator:

$$R_a = R_b \cdot \varsigma(r),$$

  where R_b is unadjusted risk and ς is a non-parametric weighting function that depends on the rank order r of the prey agent. The weighting function $\varsigma(r)$ is in turn calculated as a nonlinear nonparametric function defined by the grid arrays:

  - `predator_risk_group_dilution_abscissa`
  - `commondata::predator_risk_group_dilution_ordinate`.

In this way, the first prey agent (closest distance) is subject to the predation risk equal to the baseline unadjusted risk, whereas the last prey agent (furtherest from the predator in the group) has fully diluted predation risk equal to zero. The agent with the middle rank in the group has the adjusted risk somewhere in between the unadjusted risk and null. Thus, the predator confusion and dilution effects are relative and depend on the agent's position with respect to the predator.

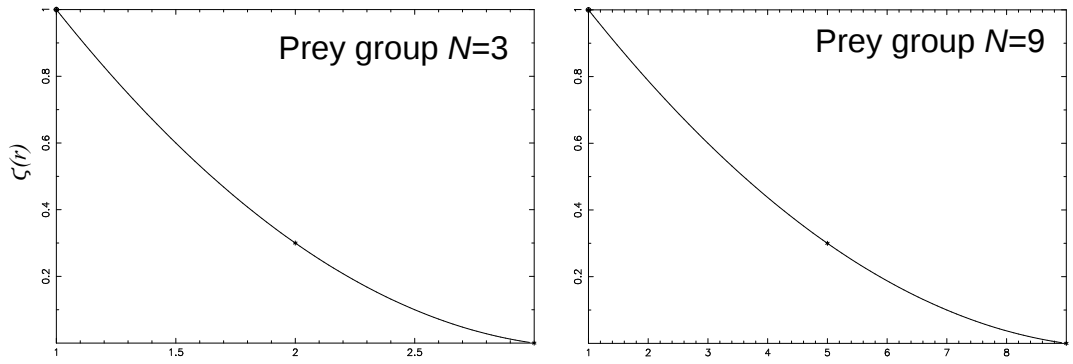


Figure 10.2 Nonparametric predator confusion/dilution factor

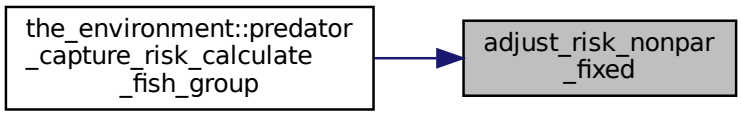
Nonetheless, the pattern is independent on the specific number of prey agents in the group, e.g. it is the same for 3 and 9 agents.
Interpolation plots can be saved in the debug mode using the command: `commondata::debug_interpolate_plot_save()`

Warning

Involves **huge** number of plots, should normally be disabled.

This is **disabled** (commented out) here to allow parallel `do concurrent` construction. If debug plots are enabled, `do concurrent` has to be altered to normal `do`.

Note that the adjusted risk is calculated only for a small subarray within the potentially huge input array of spatial prey agents, commondata::predator_risk_group_select_index_partial maximum elements. All other values are nulls.
Definition at line 10199 of file m_env.f90.
Here is the caller graph for this function:



10.4.2.10 adjust_risk_dilute_nofirst()

`subroutine predator_capture_risk_calculate_fish_group::adjust_risk_dilute_nofirst`
Adjust the predation risk of a group of *N* prey agents for predator dilution effect.
In this version, the nearest agent has a high risk equal to the baseline unadjusted risk whereas the other agents in the group have the risk diluted, on average, by their total number *N-1* (the nearest agent is excluded). The risks of the non-nearest agents still depend on the individual rank order. Whereas the average adjusted risk for all non-rank-1 agents is equal to *1/(N-1)*, each of the agents has specific risk. For the rank-2 agent it is $R_b \cdot 2/(N-1)$ while for the furtherest agent the risk is zero: $R_b \cdot 0.0$.
See the main container procedure `the_environment::predator_capture_risk_calculate_fish_group()`.

10.4.2.10.1 Notable variables

- **predator_risk_dilution** is the predator dilution weighting factor array. It is selected such that the average value over the whole array is *1/(N-1)* where *N* is the total number of prey agents in the group including the nearest agent.

10.4.2.10.2 Implementation details Calculate the array of dilution factor that is equally linearly spaced from 2/(N-1) to 0.0 (the furtherest agent). The average dilution factor for this group is therefore *1/(N-1)*.
Nonetheless, if there is only one prey agent, its risk is calculated as the full baseline risk R_b.
If, on the other hand, only two prey agents are visible to the predator, the first (nearest, rank 1) gets the baseline risk R_b and the second $R_b/2.0$.
The `LINSPACE()` procedure for generating linearly equally spaced array from HEDTOOLS is used for calculation of the dilution factor array `predator_risk_dilution`.

- The adjusted risk is equal to the baseline risk for the prey agent that has the rank one, i.e. the closest to the predator.
- However, for all other agents in the group the adjusted risk is diluted by the remaining group size (i.e. excluding the nearest agent **N-1**).

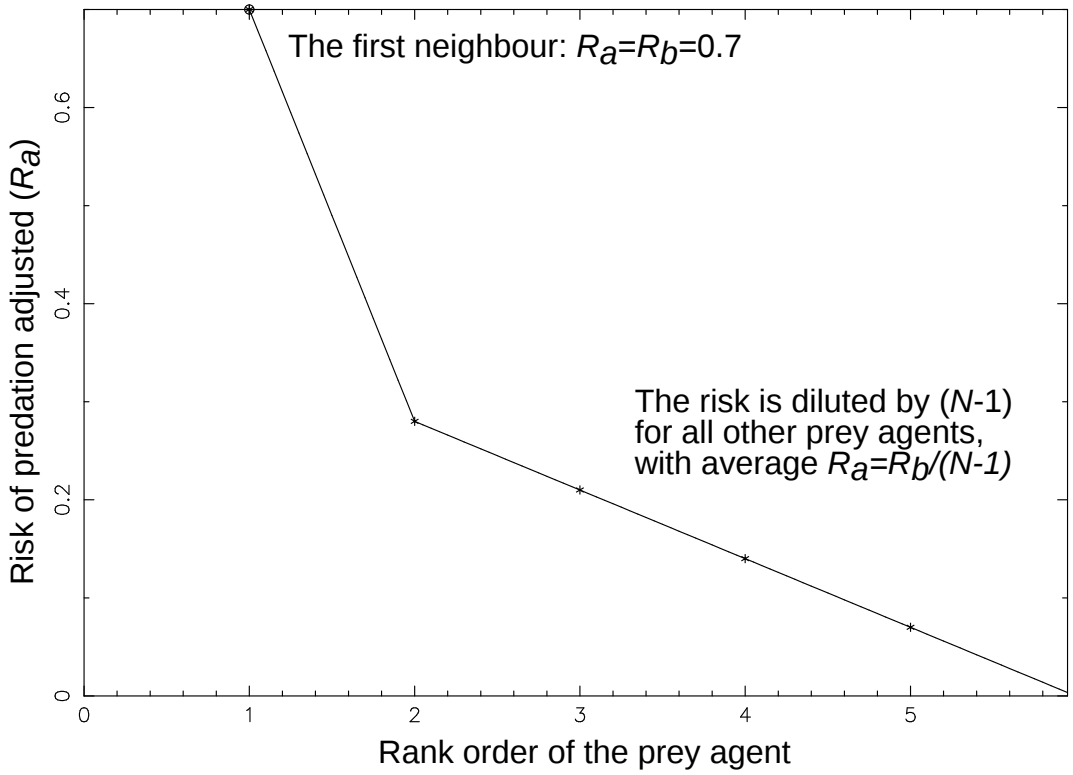


Figure 10.3 Adjustment of the predation risk by dilution factor

In the example plot above, the baseline risk for the nearest prey agent in a group of 6 is 0.7, it is unchanged. But the risk is diluted for all the remaining (rank>1) prey agents on average by *6-1=5*, even though it is still dependent on their rank by a linearly evenly spaced dilution factor `predator_risk_dilution` (the plot also for simplicity assumes the baseline risk is also $R_b = 0.7$ and is identical for all agents; in real data, the linearly spaced factor is used as a weight for specific baseline risk values R_b, so the pattern deviates from the perfect straight line).

Definition at line 10309 of file m_env.f90.

10.4.2.11 adjust_risk_dilute_all()

`subroutine predator_capture_risk_calculate_fish_group::adjust_risk_dilute_all`
Adjust the predation risk of a group of *N* prey agents for predator dilution effect.
In this version, all prey agents in the group have the risk diluted, on average, by their total number *N*. The risks of the agents are not fixed and depend on the individual rank order. Whereas the average adjusted risk is equal to *1/(N)*, each of the agents has specific risk. For the rank-1 agent it is $R_b \cdot 2/N$ while for the furtherest agent the risk is zero: $R_b \cdot 0.0$.

See the main container procedure `the_environment::predator_capture_risk_calculate_fish_group()`.

10.4.2.11.1 Notable variables

- **predator_risk_dilution** is the predator dilution weighting factor array. It is selected such that the average value over the whole array is *1/N* where *N* is the total number of prey agents in the group including the nearest agent.

10.4.2.11.2 Implementation details Calculate the array of dilution factor that is equally linearly spaced from 2/N (nearest agent) to 0.0 (the furtherest agent). The average dilution factor for this group is therefore *1/N*.
Nonetheless, if there is only one prey agent, its risk is calculated as the full baseline risk R_b.
If, on the other hand, only two prey agents are visible to the predator, they both get the same risk equal to a half of baseline $R_b/2.0$.
The `LINSPACE()` procedure for generating linearly equally spaced array from HEDTOOLS is used for calculation of the dilution factor array `predator_risk_dilution`.

- The average adjusted risk is diluted by the group size *N*. However, individual prey agents have the adjusted risk values that are weighted by the linearly equally spaced dilution value `predator_risk_dilution`. Thus, even though individual values of the adjusted risk are ranked by the distance from the predator, their average values for the whole group is diluted by the group size *N*.

Definition at line 10395 of file m_env.f90.

10.5 m_evolut.f90 File Reference

THE_EVOLUTION Module implements the Genetic Algorithm for the AHA Model.

Modules

- module the_evolution

  Implementation of the genetic algorithm.

Functions/Subroutines

- subroutine the_evolution::init_environment_objects ()

  Initialise the environmental objects. Most of the environmental objects, such as the environment, habitats etc. are kept static throughout the model running. There are, however, patterned and stochastic changes in the environment, such as diurnal variation of the illumination level.
- integer function, public the_evolution::preevol_steps_adaptive (generation)

  Calculate the adaptive number of time steps for the fixed fitness preevolution stage of the genetic algorithm.
- subroutine, public the_evolution::preevol_steps_adaptive_save_csv (csv_file_name, is_success)

  This is a diagnostic subroutine to save the number of time steps for the adaptive GA.
- subroutine the_evolution::generations_swap ()

  Swap generation pointers between parents and offspring.
- subroutine the_evolution::selection ()

  Select reproducing agents, the best commondata::ga_reproduce_pr portion of agents.
- subroutine the_evolution::mate_reproduce ()

  Mate, reproduce and mutate.
- subroutine, public the_evolution::generations_loop_ga ()

  *This procedure implements the main **Genetic Algorithm** for evolving the agents.*
- subroutine lifecycle_preevol (active_population)

  This subroutine implements the full life cycle in a whole population of agents. It is built around the main loop `LIFECYCLE_PREEVOL_LOOP`.
- subroutine generation_stats_record_write ()

  Save generation-wise statistics. This procedure only writes a single record of data after each generation. Opening the file, definition of the file handling objects that are used here etc. are done in the upstream procedure the_evolution::generations_loop_ga().

Variables

- character(len= ∗), parameter, private the_evolution::modname = "(THE_EVOLUTION)"
- type(timer_cpu), public the_evolution::stopwatch_global

  Model-global stopwatch objects.
- type(timer_cpu), public the_evolution::stopwatch_generation
- type(timer_cpu), public the_evolution::stopwatch_op_current
- type(timer_cpu), public the_evolution::single
- type(timer_cpu), public the_evolution::operation
- type(habitat), public the_evolution::habitat_safe

  We have an environment composed of two habitats, safe and a dangerous.
- type(habitat), public the_evolution::habitat_dangerous
- type(population), target, public the_evolution::generation_one

  Here we create instances for two populations which will then serve as parents and offspring. And then we declare pointers that will point to parents and offspring.
- type(population), target, public the_evolution::generation_two
- type(population), pointer, public the_evolution::proto_parents
- type(population), pointer, public the_evolution::proto_offspring

10.5.1 Detailed Description

THE_EVOLUTION Module implements the Genetic Algorithm for the AHA Model.

Author

Sergey Budaev sergey.budaev@uib.no

Jarl Giske jarl.giske@uib.no

Date

2016-2017

10.5.2 Function/Subroutine Documentation

10.5.2.1 lifecycle_preevol()

```
subroutine generations_loop_ga::lifecycle_preevol (
            class(population), intent(inout) active_population )
```

This subroutine implements the full life cycle in a whole population of agents. It is built around the main loop LIFECYCLE_PREEVOL_LOOP.

10.5.2.1.1 Implementation details First, the ages of all agents are reset to 0 before the cycle of their life (time steps of the model).
Second, each generation is subjected to selective birth mortality by the_population::population::mortality_birth() at birth, before the first time step.

Note

Forced mean and sd values from generation 1 data. Normally must be obtained from the first generation data, with global parameters added.

Then, calculate the number of time steps for the current generation. The number of time steps is based on the adaptive algorithm implemented in the the_evolution::preevol_steps_adaptive() function.
The arrays that keep time-step wise statistics for the current generation are allocated with the above number of time steps.
Some of these arrays are integer counts.
... and they also initialised to commondata::missing value (integer arrays to commondata::unknown).
Start the main life cycle loop LIFECYCLE_PREEVOL_LOOP over all the time steps (limited by the adaptive algorithm).
Reset/update the global commondata::global_time_step_model_current.

10.5.2.1.1.1 Prepare the environment Perform the sinusoidal vertical migration of the food items, they are relocating to the depth appropriate for specific time step of the model. Food migration is done here with the the_environment::migrate_food_vertical() directly on the global array of habitats the_environment::global_habitats_available to avoid the need to synchronise the array with the habitat objects.
The average distance between the food items is reported to the log. The average distance between the food items is good to know, e.g. to compare it with the agent's random walk step size.

10.5.2.1.1.2 Habitat-specific mortality Agents are subjected to random habitat-specific mortality by calling the_population::population::mortality_habitat().

Warning

Mortality is so far disabled.

10.5.2.1.1.3 Agents do a single time step of life Perform a single step of the life cycle of the whole population of agents. The agents do this step of their life cycle in a random (or non-random) order. See the_population::population::lifecycle_step() for details.
Immediately after the time step is done, time step-wise statistics are calculated.
The habitat and food resource data are disassembled back into the original static habitat objects out of the global array the_environment::global_habitats_available. This transfers the changes in the food resources (e.g. the agents consume the food) from the global array back to the original static habitat objects. See the_environment::disassemble() procedure.
Now, the time-step-wise habitat statistics can be computed for the current time step.

10.5.2.1.1.4 Maximum rescale motivation updated The population-wise maximum motivation parameter commondata::global_rescale_maximum_motivation is updated based on the global maximum value.

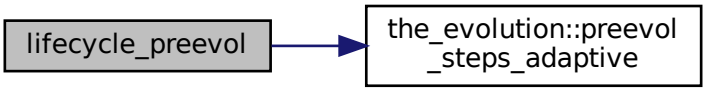
10.5.2.1.1.5 The agents are subjected to predation It is implemented by cycling over all predators within the safe and dangerous habitat and calling the the_population::population::attacked() method for each predator.

- Safe habitat: `PREDATION_HAB_SAFE` block;

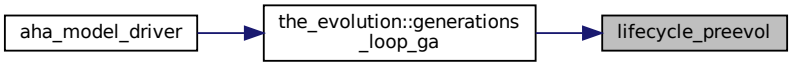
Dangerous habitat: `PREDATION_HAB_DANGER` block.

10.5.2.1.1.6 Save all agent data All agents data are saved to CSV file using the_population::population::save_csv() method. However, this is done only if the parameter commondata::enable_save_agents_each_timestep is set to TRUE. This is implemented in the `SAVE_ALL_AGENTS`block.
Additionally, the previous and the latest behaviour of the agent is also saved for each time step.
If commondata::is_zip_outputs is TRUE, CSV output data file is compressed using commondata::cmd_zip_output.

10.5.2.1.1.7 Save time-step-wise data After the life cycle loop is completed, time-step-wise statistics are saved into CSV data file for the current generation.
The CSV output data file can be optionally compressed with the commondata::cmd_zip_output command if commondata::is_zip_outputs is set to TRUE.
Definition at line 1053 of file m_evolut.f90.
Here is the call graph for this function:



Here is the caller graph for this function:



10.5.2.2 generation_stats_record_write()

`subroutine generations_loop_ga::generation_stats_record_write [private]`

Save generation-wise statistics. This procedure only writes a single record of data after each generation. Opening the file, definition of the file handling objects that are used here etc. are done in the upstream procedure the_evolution::generations_loop_ga().

Warning

This subroutine neither opens nor closes the output CSV file, only writes a single record of statistical data from the current generation commondata::global_generation_number_current into it.

10.5.2.2.1 Implementation notes First, initialise an empty record for CSV data.
Then calculate and append each of the statistical data fields to build the complete record of the CSV output file. Note that the fields must agree with the columns defined by the `FILE_STATS_GENER_COLS` parameter array.

- "GENERATION" – generation number;

"PREEVOL_STEPS" – lifespan, number of time steps;

- "MUTAT_POINT" – adaptive rate of point mutations;
- "MUTAT_BATCH" – adaptive rate of batch mutations;
- "ELITE_GROUP" – the number of reproducing agents;
- "N_ALIVE" – number of agents alive at the end;
- "N_GROWN" – number of agents that had grown;
- "N_MALES_L" – number of males alive;
- "N_FEMALES_L" number of females alive;
- "N_EATEN_PRED" number of agents that are eaten by predators;
- "BODY_MASS" – average body mass;
- "BODY_LEN" – average body length;
- "BIRTH_MASS" – body mass at birth;
- "BIRTH_LENGTH" – body length at birth;
- "BIRTH_ENERGY" – energy reserves at birth;
- "ENERGY" – energy reserve;
- "STOMACH" – stomach contents, mass;
- "SMR" – average SMR;
- "CTRL_RND" – average control trait;

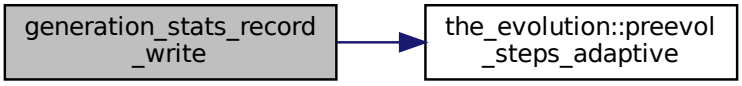
- "REPRFACT" – average reproductive factor;
- "P_REPR" – probability of reproduction;
- "N_REPROD" – total nuber or reproductions;
- "N_OFFSPRING" – number of offspring;
- "GOS_AROUSAL" – GOS arousal;
- "FOODS_TRY" – average number of attempts to catch food items;
- "FOODS_EATEN" – average number of food items eaten;
- "FMASS_EATEN" – average number of food items eaten;
- "PERC_FOOD" – food perception, average;
- "PERC_CONS" – conspecific perception, average;
- "PERC_PRED" – predator perception, average;
- "DEPTH" – location depth at the end,
- "N_SAFE_HABITAT" – number of agents in the "safe" habitat;
- "N_DANG_HABITAT" – number of agents in the "dangerous" habitat.
- Calculate perception averages in the safe habitat:
- "PERC_FOOD_SAFE" – food perception in "safe" habitat;
- "PRC_FDIST_SAFE" – food perception in "safe" habitat;
- "PERC_CONS_SAFE" – conspecific perception in "dangerous" habitat;
- "PERC_PRED_SAFE" – predator perception in "safe" habitat;
- Calculate perception averages in the dangerous habitat:
- "PERC_FOOD_DANG" – food perception in "dangerous" habitat;
- "PRC_FDIST_DANG" – food perception in "dangerous" habitat;
- "PERC_CONS_DANG" – conspecific perception in "dangerous" habitat;
- "PERC_PRED_DANG" – predator perception in "dangerous" habitat;
- "FDIST_SAFE" – average distance between food items in the safe habitat;
- "FDIST_DANGER" - average distance between food items in the dangerous habitat;
- "FITNESS_MIN" – minimum fitness value.
- "FITNESS_MEAN" – average fitness.
- "N_FOODS_SAFE" – number of food items available in "safe" habitat;
- "N_FOODS_DANG" – number of food items available in "dangerous" habitat.

The following characteristics are calculated for **alive** agents.
Here the `ALIVE` block implements sorting out the individuals that are the_genome:individual_genome::is_alive().

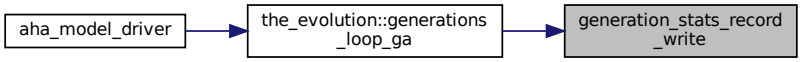
- "BODY_MASS_L" – average body mass of alive agents;
- "BODY_LENGTH_L" – average body length of alive agents;
- "ENERGY_L" – average energy reserves of alive agents;
- "SMR_L" – average SMR of alive agents;

- "CONTROL_L" – control trait of alive agents;
- "REPRFACT_L" – reproductive factor of alive agents;
- "P_REPROD_L" – probability of reproduction of alive agents;
- "FOODS_TRY_L" – average rate of attempts to catch food items by alive agents;
- "FOODS_EATEN_L" – average rate of successful food captures;
- "FMASS_EATEN_L" – average rate of successful food captures;
- "N_SAFE_HAB_L" – number of alive agents in "safe" habitats;
- "N_DANG_HAB_L" – number of alive agents in "dangerous" habitat;
- "FITNESS_MEAN_L" – mean fitness of alive agents.

Once the record is fully built, it is written to the file using the `CSV_RECORD_WRITE` procedure (see `CSV_IO`).
Definition at line 1425 of file m_evolut.f90.
Here is the call graph for this function:



Here is the caller graph for this function:



10.6 m_fileio.f90 File Reference

File objects for the AHA Model.

Data Types

- type file_io::file_handle

  `FILE_HANDLE` is the basic file handle object. It provides an unitary object oriented interface for operations with any supported file types.

Modules

- module file_io

  Definition of high level file objects.

Functions/Subroutines

- logical function file_io::file_operation_last_is_success (this)

  Get the success or error status of the latest file operation. ***Example:***
- character(len=:) function, allocatable file_io::file_hangle_get_name_string (this)

  Get the file name associated with the file handle. If the file name is (yet) undefined, the latest operation success flag (see file_io::is_success()) is FALSE. ***Example:***
- integer function file_io::file_object_get_associated_unit (this)

  A Low level function to get the Fortran unit number associated with the file handle object.
- logical function file_io::file_object_format_is_csv (this)

  Check if the file format is CSV.
- logical function file_io::file_object_format_is_txt (this)

  Check if the file format is CSV.
- subroutine file_io::csv_open_write_this (this, name, format)

  This is an object oriented wrapper for `CSV_OPEN_WRITE()`*. For details see* `CSV_OPEN_WRITE`*.*
- subroutine file_io::csv_close_this (this)

  This is an object oriented wrapper for `CSV_CLOSE()`*. For details see* `CSV_CLOSE`*.*
- subroutine file_io::csv_header_line_write_this (this, header)

  This is an object oriented wrapper for `CSV_HEADER_WRITE()`*. See* `CSV_HEADER_WRITE` *for details.*
- subroutine file_io::csv_record_string_write_this (this, csv_record)

  Physically write a single string CSV data record to the file. See `CSV_RECORD_WRITE` ***Example:***

Public enumeration constants defining supported file types

Define the file types that are supported by this module.

- enum
- @, public file_io::undefined
- @, public file_io::format_csv
- @, public file_io::format_txt

10.6.1 Detailed Description

File objects for the AHA Model.

Author

Sergey Budaev `sergey.budaev@uib.no`

Jarl Giske `jarl.giske@uib.no`

Date

2016-2017

10.7 m_genome.f90 File Reference

The Genome objects of the AHA Model.

Data Types

- type the_genome::gene

  This describes an individual gene object. See the genome structure for as general description and gene for details.
- type the_genome::chromosome

  This type describes the chromosome object. Chromosome consists of an array of alleles and a descriptive string label. See "the genome structure" for as general description and "chromosome" for details.
- type the_genome::individual_genome

  This type describes parameters of the individual agent's genome The genome is an array of allocatable the_genome::chromosome objects, different kinds of agents may have different genomes with different number of chromosomes. See "the genome structure" for as general description and "genome" for details.

Modules

- module the_genome

  Definition the genetic architecture of the agent.

Functions/Subroutines

- elemental subroutine the_genome::chromosome_sort_rank_id (this)

  Sort GENE objects within the CHROMOSOME by their `rank_id`. The two subroutines `qsort` and `qs_partition_rank_id` are a variant of the recursive quick-sort algorithm adapted for sorting integer components of the the `CHROMOSOME` object.
- recursive pure subroutine qsort (A)

  `qsort` and `qs_partition_` are the two parts of the recursive sort algorithm `qsort` is the recursive frontend. Sorts genes within the chromosome by components of the array of `alleles`.
- pure subroutine qs_partition_rank_id (A, marker)

  `qsort` and `qs_partition_` are the two parts of the recursive sort algorithm `qs_partition_rank_id` is a pivot backend, here it sorts genes within the chromosome object by integer `rank_id` components of the genes.
- subroutine the_genome::allele_init_random (this)

  *Initialises allele with a random integer. Note that we do **not** set the labels for the alleles here during the random initialisation.*
- elemental subroutine the_genome::allele_create_zero (this)

  Create allele with zero value. We don't set labels for alleles here.
- subroutine the_genome::allele_label_init_random (this)

  The (pair of) alleles here are assigned random string labels Not sure if that is necessary for any application though.
- elemental subroutine the_genome::allele_label_set (this, label)

  Set labels for the alleles. The subroutine parameter is array of labels.
- elemental character(len=label_length) function the_genome::allele_label_get (this)

  Get the i-th allele label.
- elemental subroutine the_genome::allele_value_set (this, set_value, nr)

  Set a single value of the allele additive component.
- pure subroutine the_genome::alleles_value_vector_set (this, values)

  Set values of the alleles as a vector, i.e. sets the whole gene values.
- elemental integer function the_genome::allele_value_get (this, nr)

  Get the value of the allele.
- pure subroutine the_genome::allele_values_vector_get (this, values)

  Get the vector of all values of the alleles, i.e. gets the gene values.
- elemental subroutine the_genome::allele_rank_id_set (this, value_id)
- subroutine the_genome::allele_mutate_random (this, prob)

  Introduce a random point mutation to a random allele component.
- subroutine the_genome::allele_mutate_random_batch (this, prob)

  Introduce a random mutation of the whole set of additive allele components.
- subroutine the_genome::chromosome_init_allocate_random (this, length, label)

  This subroutine initialises the chromosome with, and allocates, random alleles, sets one of them randomly dominant and optionally defines the chromosome label.
- subroutine the_genome::chromosome_create_allocate_zero (this, length, label)

  Init a new chromosome, zero, non-random.
- elemental subroutine the_genome::chromosome_recalculate_rank_ids (this)

  This subroutine recalculates rank_id indices for consecutive gene objects within the chromosome. This may be necessary after reordering by random relocation mutation.
- subroutine the_genome::chromosome_mutate_relocate_swap_random (this, prob)

  *Mutate within the same chromosome, relocate a gene (unit of alleles) to a different random position within the same chromosome, the misplaced gene moves to the relocated gene position, so they are just **swap**.*
- subroutine the_genome::chromosome_mutate_relocate_shift_random (this, prob)

*Mutate within the same chromosome, relocate a gene (unit of alleles) to a different random position within the same chromosome, **shifting** all other genes within the chromosome down one position. This works as follows: first, we randomly determine the gene to relocate, assign it a new random rank_id. Then re-sort the chromosome according to the new ranks with `qsort` with the `qs_partition_rank_id` backend.*

- subroutine the_genome::genome_init_random (this, label)

  Initialise the genome at random, and set sex as determined by the sex determination locus.
- subroutine the_genome::genome_create_zero (this)

  Create a new empty genome, and set sex as determined by the sex determination locus. Genome values are from parents using inherit functions.
- subroutine the_genome::genome_label_set (this, label)

  Label genome. If label is not provided, make a random string.
- elemental character(len=label_length) function the_genome::genome_label_get (this)

  Accessor function to get the genome label. The label is a kind of a (random) text string name of the genome and the individual agent.
- subroutine the_genome::genome_sex_determine_init (this)

  Sex determination initialisation subroutine.
- elemental logical function the_genome::genome_get_sex_is_male (this)

  Get the logical sex ID of the genome object component.
- elemental logical function the_genome::genome_get_sex_is_female (this)

  Get the logical sex ID of the genome object component.
- elemental character(len=label_length) function the_genome::genome_get_sex_label (this)

  Get the descriptive sex label: male or female.
- elemental subroutine the_genome::genome_individual_set_alive (this)

  *Set the individual to be **alive**, normally this function is used after init or birth.*
- elemental subroutine the_genome::genome_individual_set_dead (this)

  *Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function.*
- elemental logical function the_genome::genome_individual_check_alive (this)

  *Check if the individual is **alive**.*
- elemental logical function the_genome::genome_individual_check_dead (this)

  *Check if the individual is **dead** (the opposite of `is_alive`).*
- subroutine the_genome::genome_individual_recombine_homol_full_rand_alleles (this, mother, father, exchange_ratio)

  Internal genetic recombination backend, exchange individual alleles between homologous chromosomes in mother and father genomes to form the `this` (offspring) genome. Fully random recombination.
- subroutine the_genome::genome_individual_recombine_homol_part_rand_alleles (this, mother, father, exchange_ratio)

  Internal genetic recombination backend, exchange individual alleles between homologous chromosomes in mother and father genomes to form the `this` (offspring) genome. Partially random recombination, identical across the homologous chromosomes.
- subroutine the_genome::genome_individual_crossover_homol_fix (this, mother, father, pattern_matrix)

  Internal fixed genetic crossover backend, exchange blocks of alleles between homologous chromosomes in mother and father genomes to form the `this` (offspring) genome.
- subroutine the_genome::genome_mutate_wrapper (this, p_point, p_set, p_swap, p_shift)

  Perform a probabilistic random mutation(s) on the individual genome. This is a high level wrapper to build mutations from various components.

Neuronal response functions

There are two separate functions that produce a trait value from the genotype. The procedure `trait_init_genotype_gamma2gene` does modify the agent (`this`, intent[inout]) as it sets the label. On the other hand, a similar procedure `the_genome::trait_set_genotype_gamma2gene()` does not affect the agent, it has the intent [in].

- subroutine the_genome::trait_init_genotype_gamma2gene (this, this_trait, g_p_matrix, init_val, gerror_cv, label)

*Initialise an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy.*

- subroutine the_genome::trait_set_genotype_gamma2gene (this, this_trait, g_p_matrix, init_val, gerror_cv)

  *Set an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy.*

The genotype to phenotype functions based on fixed linear

transformation.

- subroutine the_genome::trait_init_linear_sum_additive_comps_2genes_r (this, this_trait, g_p_matrix, phenotype_min, phenotype_max, label)

  *Initialise an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy.*
- subroutine the_genome::trait_set_linear_sum_additive_comps_2genes_r (this, this_trait, g_p_matrix, phenotype_min, phenotype_max)

  *Set an **individual trait** of the agent that depends on the genotype. This can be any trait upwards in the class hierarchy.*

Variables

- character(len= ∗), parameter, private the_genome::modname = "(THE_GENOME)"

10.7.1 Detailed Description

The Genome objects of the AHA Model.

Author

Sergey Budaev sergey.budaev@uib.no

Jarl Giske jarl.giske@uib.no

Date

2016-2017

10.7.2 Function/Subroutine Documentation

10.7.2.1 qsort()

```
recursive pure subroutine chromosome_sort_rank_id::qsort (
            type(gene), dimension(:), intent(inout) A )
```

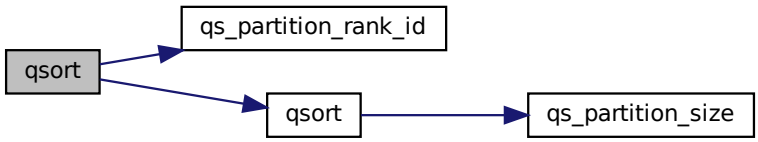
`qsort` and `qs_partition_` are the two parts of the recursive sort algorithm `qsort` is the recursive frontend. Sorts genes within the chromosome by components of the array of `alleles`.

Parameters

`in,out`	*a*	[inout] input array to be sorted. It has the same type as the individual component objects of the array-object that we are going to sort.

Definition at line 334 of file m_genome.f90.

Here is the call graph for this function:



10.7.2.2 qs_partition_rank_id()

```
pure subroutine chromosome_sort_rank_id::qs_partition_rank_id (
            type(gene), dimension(:), intent(inout) A,
            integer, intent(out) marker )
```

`qsort` and `qs_partition_` are the two parts of the recursive sort algorithm `qs_partition_rank_id` is a pivot backend, here it sorts genes within the chromosome object by integer `rank_id` components of the genes.

Parameters

`in,out`	*a*	[inout] input array to be sorted.
`out`	*marker*	internal pivot marker.

Note

Pivot point `x`, has the same type **as the sorted object component**.

Definition at line 355 of file m_genome.f90.
Here is the caller graph for this function:

10.8 m_hormon.f90 File Reference

The Hormone architecture of the AHA Model.

Data Types

- type the_hormones::hormones

  This type adds hormonal architecture extending the genome object.

Modules

- module the_hormones

Definition the hormonal architecture of the agent.

Functions/Subroutines

- subroutine the_hormones::hormones_init_genotype (this)

  Initialise hormone levels based on the genome value. Two alleles are selected at random and input into the `gamma2gene` *function to get the initial hormone values rescaled to 0:1. Note that the* `gamma2gene` *alleles defining the* ***shape*** *of the gamma function and the* ***half-max effect*** *are selected randomly in this version. Also, polyploid organisms are possible, in such case, two parameters are also randomly defined from a larger set (e.g. from four chromosomes in case of tetraploids). See implementation details and comments for each of the hormones.*
- elemental subroutine the_hormones::hormones_clean_history_stack (this)

  Clean the history stack of hormones: testosterone and estrogen histories are set to `MISSING`*.*
- elemental subroutine the_hormones::hormones_update_history (this)

  Update the sex steroid hormones history stack from the current level.
- elemental real(srp) function the_hormones::hormones_reproductive_factor_calc (this)

  Calculate the reproductive factor. Reproductive factor is defined as the current level of the_hormones::testosterone_level in males and the_hormones::estrogen_level in females.
- elemental real(srp) function the_hormones::testosteron_baseline_get_level (this)

  Get the value of testosterone baseline.
- elemental real(srp) function the_hormones::estrogen_baseline_get_level (this)

  Get the value of estrogen baseline.

Accessor functions for all the hormones.

Get and set functions for each hormone follow. We left them as individual hormone-specific functions duplicating code. Not ideal, but easy to use provided hormones do not change too often. Tiny atomic hormone get/set functions are easy to code.

- elemental real(srp) function the_hormones::growhorm_get_level (this)

  Get the value of ***growth hormone****.*
- elemental subroutine the_hormones::growhorm_set_level (this, value_set)

  Set the value of ***growth hormone****.*
- elemental real(srp) function the_hormones::thyroid_get_level (this)

  Get the value of ***thyroid****.*
- elemental subroutine the_hormones::thyroid_set_level (this, value_set)

  Set the value of ***thyroid****.*
- elemental real(srp) function the_hormones::adrenaline_get_level (this)

  Get the value of ***adrenaline****.*
- elemental subroutine the_hormones::adrenaline_set_level (this, value_set)

  Set the value of ***adrenaline****.*
- elemental real(srp) function the_hormones::cortisol_get_level (this)

  Get the value of ***cortisol****.*
- elemental subroutine the_hormones::cortisol_set_level (this, value_set)

  Set the value of ***cortisol****.*
- elemental real(srp) function the_hormones::testosterone_get_level (this)

  Get the value of ***testosterone****.*
- elemental subroutine the_hormones::testosterone_set_level (this, value_set, update_history)

  Set the value of ***testosterone****.*
- elemental real(srp) function the_hormones::estrogen_get_level (this)

  Get the value of ***estrogen****.*
- elemental subroutine the_hormones::estrogen_set_level (this, value_set, update_history)

  Set the value of ***estrogen****.*

Variables

- character(len= $*$), parameter, private the_hormones::modname = "(THE_HORMONES)"

10.8.1 Detailed Description

The Hormone architecture of the AHA Model.

Author

Sergey Budaev sergey.budaev@uib.no
Jarl Giske jarl.giske@uib.no

Date

2016-2017

10.9 m_indiv.f90 File Reference

Definition of the individual agent in the AHA Model.

Data Types

- type the_individual::individual_agent

  This type describes parameters of the individual agent.

Modules

- module the_individual

  An umbrella module that collects all the components of the individual agent.

Functions/Subroutines

- subroutine, private the_individual::individual_init_random (this, exclude_genome)

  Generate a random agent from the genotype.
- subroutine the_individual::individual_create_zero (this)

  Generate a new empty agent.
- elemental subroutine the_individual::individual_agent_set_dead (this)

  *Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function.*
- subroutine the_individual::kill_destroy (this)

  Finalisation procedure. Note that finalisation of objects may not yet be implemented in the compiler. Therefore this subroutine is not used so far, just a stub.
- elemental real(srp) function the_individual::individual_get_risk_mortality_individual (this)

  Get the individually-specific mortality risk for the agent.
- elemental subroutine the_individual::individual_preevol_fitness_calc (this)

  Calculate fitness for the pre-evolution phase of the genetic algorithm. Pre-evolution is based on selection for a simple criterion without explicit reproduction etc. The criterion for selection at this phase is set by the integer the_individual::individual_agent::fitness component.
- elemental integer function fitness_birth_mass_ratio ()

  Fitness is calculated as the ratio of the birth mass to the current mass. This value is weighted by the multiplier (1000) to get a fairly large integer (so decimals are unimportant) and also weighted by the number of food items eaten and the number of offspring produced.
- elemental integer function fitness_stomach_mass_ratio ()

  Fitness is calculated as the ratio of the body mass to the stomach content.
- elemental integer function fitness_stomach_mass_abs ()

  Fitness is calculated as the absolute stomach content.
- elemental integer function fitness_food_mass_sum ()

  Fitness is calculated as the cumulative mass of all food objects eaten.
- elemental integer function fitness_mass_incr_ratio ()

Fitness is calculated as the mass increment in units of birth mass.

- elemental integer function fitness_mass_incr_abs ()

  Fitness is calculated as absolute mass increment from the birth mass.

- elemental integer function fitness_reprod_factor ()

  Fitness as the reproductive factor.

- elemental integer function fitness_energy_reprfact ()

  Fitness as a weighted combination of energy and reproductive factor.

- elemental integer function fitness_energy_reprfact_mass ()

  Fitness as a weighted combination of mass increment, energy, and reproductive factor.

Variables

- character(len= ∗), parameter, private the_individual::modname = "(THE_INDIVIDUAL)"

10.9.1 Detailed Description

Definition of the individual agent in the AHA Model.

Author

Sergey Budaev sergey.budaev@uib.no

Jarl Giske jarl.giske@uib.no

Date

2016-2017

10.9.2 Function/Subroutine Documentation

10.9.2.1 fitness_birth_mass_ratio()

`elemental integer function individual_preevol_fitness_calc::fitness_birth_mass_ratio`

Fitness is calculated as the ratio of the birth mass to the current mass. This value is weighted by the multiplier (1000) to get a fairly large integer (so decimals are unimportant) and also weighted by the number of food items eaten and the number of offspring produced.

Note

This procedure is internal to the_individual::individual_agent::fitness_calc().

Returns

Fitness value.

`INT_MULTIPLIER_FITNESS` is a multiplier to set the appropriate scaling for the initial the_individual::individual_agent::fitness.
`INT_WEIGHT_FEEDING` is an integer weight given to the any non-zero successful feeding.
`INT_WEIGHT_OFFSPRING` is an integer weight given to the any non-zero successful reproductions
Initial fitness is the ratio of the birth mass to the current mass weighted by the `INT_MULTIPLIER_FITNESS`.
If the agent successfully caught and eaten any number of food items, its fitness is divided by the number of food items eaten weighted by the `INT_WEIGHT_FEEDING` parameter.
If the agent has successfully done any reproduction, its fitness is divided by the number of offspring weighted by `INT_WEIGHT_OFFSPRING`.
Definition at line 283 of file m_indiv.f90.

10.9.2.2 fitness_stomach_mass_ratio()

`elemental integer function individual_preevol_fitness_calc::fitness_stomach_mass_ratio`
Fitness is calculated as the ratio of the body mass to the stomach content.

Note

This procedure is internal to the_individual::individual_agent::fitness_calc().

Returns

Fitness value.

`INT_MULTIPLIER_FITNESS` is a multiplier to set the appropriate scaling for the initial the_individual::individual_agent::fitness.
Definition at line 325 of file m_indiv.f90.

10.9.2.3 fitness_stomach_mass_abs()

`elemental integer function individual_preevol_fitness_calc::fitness_stomach_mass_abs`
Fitness is calculated as the absolute stomach content.

Note

This procedure is internal to the_individual::individual_agent::fitness_calc().

Returns

Fitness value.

`INT_MULTIPLIER_FITNESS` is a multiplier to set the appropriate scaling for the initial the_individual::individual_agent::fitness.
Definition at line 347 of file m_indiv.f90.

10.9.2.4 fitness_food_mass_sum()

`elemental integer function individual_preevol_fitness_calc::fitness_food_mass_sum`
Fitness is calculated as the cumulative mass of all food objects eaten.

Note

This procedure is internal to the_individual::individual_agent::fitness_calc().

Returns

Fitness value.

`INT_MULTIPLIER_FITNESS` is a multiplier to set the appropriate scaling for the initial the_individual::individual_agent::fitness.
Definition at line 369 of file m_indiv.f90.

10.9.2.5 fitness_mass_incr_ratio()

`elemental integer function individual_preevol_fitness_calc::fitness_mass_incr_ratio`
Fitness is calculated as the mass increment in units of birth mass.

Note

This procedure is internal to the_individual::individual_agent::fitness_calc().

Returns

Fitness value.

`INT_MULTIPLIER_FITNESS` is a multiplier to set the appropriate scaling for the initial the_individual::individual_agent::fitness.
Definition at line 391 of file m_indiv.f90.

10.9.2.6 fitness_mass_incr_abs()

`elemental integer function individual_preevol_fitness_calc::fitness_mass_incr_abs`
Fitness is calculated as absolute mass increment from the birth mass.

Note

This procedure is internal to the_individual::individual_agent::fitness_calc().

Returns

Fitness value.

`INT_MULTIPLIER_FITNESS` is a multiplier to set the appropriate scaling for the initial the_individual::individual_agent::fitness.
Definition at line 413 of file m_indiv.f90.

10.9.2.7 fitness_reprod_factor()

`elemental integer function individual_preevol_fitness_calc::fitness_reprod_factor`
Fitness as the reproductive factor.

Note

This procedure is internal to the_individual::individual_agent::fitness_calc().

Returns

Fitness value.

`INT_MULTIPLIER_FITNESS` is a multiplier to set the appropriate scaling for the initial the_individual::individual_agent::fitness.
Definition at line 435 of file m_indiv.f90.

10.9.2.8 fitness_energy_reprfact()

`elemental integer function individual_preevol_fitness_calc::fitness_energy_reprfact`
Fitness as a weighted combination of energy and reproductive factor.

Note

This procedure is internal to the_individual::individual_agent::fitness_calc().

Returns

Fitness value.

`INT_MULTIPLIER_FITNESS` is a multiplier to set the appropriate scaling for the initial the_individual::individual_agent::fitness.
`FITNESS_WEIGHT_ENERGY` is the weighting factor for energy, respective weighting for the reproductive factor is then `1 - FITNESS_WEIGHT_ENERGY`.
Fitness of an alive agent cannot be smaller than in a dead agent.
Definition at line 457 of file m_indiv.f90.

10.9.2.9 fitness_energy_reprfact_mass()

`elemental integer function individual_preevol_fitness_calc::fitness_energy_reprfact_mass`
Fitness as a weighted combination of mass increment, energy, and reproductive factor.

Note

This procedure is internal to the_individual::individual_agent::fitness_calc().

Returns

Fitness value.

- `INT_MULTIPLIER_FITNESS` is a multiplier to set the appropriate scaling for the initial the_individual::individual_agent::fitnes
- `FITNESS_WEIGHT_MASINC` is the weighting factor for relative weight increment.
  - `FITNESS_WEIGHT_ENERGY` is the weighting factor for energy,

Note

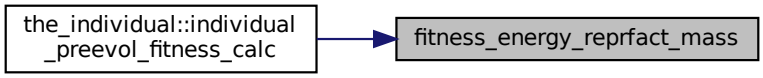
their sum must not exceed 1.0

If mass increment is negative, fitness is given a big value **1000,000**,

Note

Note that this should be smaller than the fitness of a dead agent defined in commondata::ga_fitness_dead .

Then fitness function is calculated as a weighted sum of relative weight increment W1 (M-Mb)/Mb + W2 E + W3 Rf
Fitness of an alive agent is always limited to lay within the range from 0 to fitness of a dead agent.
Definition at line 488 of file m_indiv.f90.
Here is the caller graph for this function:



10.10 m_neuro.f90 File Reference

The Neurobiological and behaviour architecture of the AHA Model.

Data Types

- type the_neurobio::percept_food

  This type defines how the agent perceives food items. The food perception object the_neurobio::percept_food is basically an array of food objects within the visual range of the agent plus distances to the agent. This is the "objective" perception container, reflecting the "real world". We introduce a perception error when perception object is analysed by the agent's neurobiological system.
- type the_neurobio::spatial_percept_component

  This type defines a single spatial perception component, i.e. some single elementary spatial object that can be perceived by the agent from a big array of objects of the same type which are available in the agent's environment. Different kinds of perception objects (e.g. conspecifics, predators etc.) can be produced by extending this basic type.
- type the_neurobio::conspec_percept_comp

  *This type defines a **single conspecific** perception component. It is required for the `the_neurobio::percept_conspecifics` type that defines the whole conspecifics perception object (array of conspecifics).*
- type the_neurobio::percept_conspecifics

  This type defines how the agent perceives conspecifics.
- type the_neurobio::spatialobj_percept_comp

  *This type defines a **single** arbitrary spatial object perception component. For example, a predator perception object is then an array of such spatial object perception components.*
- type the_neurobio::percept_predator

  This type defines how the agent perceives a predator.

- type the_neurobio::percept_stomach

  This type defines how the agent perceives its own stomach capacity.
- type the_neurobio::percept_body_mass

  This type defines how the agent perceives its own body mass it can be important for state-dependency.
- type the_neurobio::percept_energy

  This type defines how the agent perceives its own energy reserves it can be important for state-dependency.
- type the_neurobio::percept_age

  This type defines how the agent perceives its own age in terms of the model discrete time step.
- type the_neurobio::percept_reprfact

  Perception of the reproductive factor, reproductive factor depends on the sex hormones differently in males and females.
- type the_neurobio::percept_light

  Perception of the ambient illumination. This is a very simple perception component, singular and static.
- type the_neurobio::percept_depth

  Perception of the current depth horizon.
- type the_neurobio::memory_perceptual

  Individual perception memory(history) stack, a memory component that saves perception values at previous time steps of the model. Not whole perception objects are saved for simplicity, only the most important parameters, integer and real types so commondata::add_to_history() can be used in unmodified form. Decision making can make use of this memory stack.
- type the_neurobio::perception

  *The perception architecture of the agent. See "The perception mechanism" for a general overview. At this level, lower order perception objects are combined into the the_neurobio::perception class hierarchy level of the agent. The object bound functions see_ and feel_ obtain (**set**) the specific perception objects from the external or internal environments of the agent and put them into the the_neurobio::perception data structure. Also, memory component is updated with the perception data. Perception objects can then be used as input into the individual decision-making procedures.*
- type the_neurobio::percept_components_motiv

  Perceptual components of motivational states. Plugged into all `STATE_`, attention etc. These components are linked to specific inner or outer perception objects (stimuli). Their sum result(s) in the overall value of the motivation component.
- type the_neurobio::state_motivation_base

  *These types describe the **neurobiological states** of the agent. (1) Each state may have several components that are related to specific inner or outer perception objects (stimuli). (2) There is also a `motivation` component that describes the global **motivation** value for this state.*
- interface the_neurobio::motivation_init_root

  *Abstract interface for the deferred **init** function `clean_init` that has to be overridden by each object that extends the basic motivational state type.*
- type the_neurobio::state_hunger

  *The motivational state of **hunger**. Evokes food seeking, eating, higher activity, emigrating and habitat switching.*
- type the_neurobio::state_fear_defence

  *The state of **fear state**. Evokes active escape, fleeing, emigration and habitat switch.*
- type the_neurobio::state_reproduce

  *The state of **reproduction**. Evokes seeking conspecifics and mating during the reproductive phase.*
- type the_neurobio::motivation

  ***Motivation** is a collection of all internal motivational states of the agent. This type is also used in defining Expectancies of motivations.*
- type the_neurobio::memory_emotional

  *Individual motivation/emotion memory stack, a memory component that saves the values of the **final motivations** at previous time steps of the model. Not whole state (`STATE_`) objects are saved for simplicity. `add_to_history` is used in unmodified form. Decision making can make use of this emotional memory stack.*
- type the_neurobio::appraisal

  *The **appraisal** level. At this level, perception objects are feed into the commondata::gamma2gene() sigmoid function and the neuronal responses are obtained at the output. Neuronal responses for different perception objects are then summed up and the promary motivation values are obtained. Following this, modulation alters some of the primary motivation values resulting in the final motivation values. See "From perception to GOS" for an overview.*

- type the_neurobio::gos_global

  Global organismic state (GOS) level. GOS is defined by the dominant motivational state component (`STATE_`), namely, by the logical flag %dominant_state. If this logical flag is TRUE for a particular motivational state component, this state is the GOS. Thus, there should be is no separate data component(s) e.g. "value" for GOS. The values the_neurobio::gos_global::gos_main and the_neurobio::gos_global::gos_arousal can be inferred from the motivations, here are doubled mainly for convenience. See "From perception to GOS" for an overview.

Modules

- module the_neurobio

  Definition of the decision making and behavioural the architecture.

Functions/Subroutines

- elemental subroutine the_neurobio::percept_food_create_init (this, maximum_number_food_items)

  *Initiate an empty **food** perception object with known number of components.*
- subroutine the_neurobio::percept_food_number_seen (this, number_set)

  Set the total number of food items perceived (seen) in the food perception object. Do not reallocate the perception object components with respect to this new number yet.
- subroutine the_neurobio::percept_food_make_fill_arrays (this, items, dist)

  Make the food perception object, fill it with the actual data arrays.
- elemental integer function the_neurobio::percept_food_get_count_found (this)

  Get the number (count) of food items seen. Trivial.
- elemental real(srp) function the_neurobio::percept_food_get_meansize_found (this)

  Get the average size of food items seen. Trivial.
- elemental real(srp) function the_neurobio::percept_food_get_meanmass_found (this)

  Get the average mass of food items seen. Trivial.
- elemental real(srp) function the_neurobio::percept_food_get_meandist_found (this)

  Get the average distance to the food items seen. Trivial.
- elemental subroutine the_neurobio::percept_food_destroy_deallocate (this)

  *Deallocate and delete a **food** perception object.*
- subroutine the_neurobio::food_perception_get_visrange_objects (this, food_resource_available, time_step↩_model)

  Get available food items within the visual range of the agent, which the agent can perceive and therefore respond to. Food perception is packaged into the food perception object this%perceive_food for output.
- elemental logical function the_neurobio::food_perception_is_seeing_food (this)

  Check if the agent sees any food items within its visual range.
- real(srp) function the_neurobio::food_perception_probability_capture_memory_object (this, last, time_step↩_model)

  Calculate the probability of capture of a subjective representation of food item based on the data from the perceptual memory stack.
- elemental subroutine the_neurobio::percept_stomach_create_init (this)

  *Initiate an empty **stomach** capacity perception object.*
- elemental real(srp) function the_neurobio::percept_stomach_get_avail_capacity (this)

  *Get the currently available value of the available **stomach** volume.*
- subroutine the_neurobio::percept_stomach_update_avail_capacity (this, current_volume)

  *Set and update the currently available value of the available **stomach** volume.*
- elemental subroutine the_neurobio::percept_stomach_destroy_deallocate (this)

  *Destroy the **stomach** perception object and deallocate it.*
- elemental subroutine the_neurobio::percept_bodymass_create_init (this)

  *Initiate an empty **body mass** perception object.*
- elemental real(srp) function the_neurobio::percept_bodymass_get_current (this)

  *Get the current value of the **body mass** perception.*

- subroutine the_neurobio::percept_bodymass_update_current (this, current)

  *Set and update the current **body mass** perception value.*
- elemental subroutine the_neurobio::percept_bodymass_destroy_deallocate (this)

  *Destroy the **body mass** perception object and deallocate.*
- elemental subroutine the_neurobio::percept_energy_create_init (this)

  *Initiate an empty **energy** perception object.*
- elemental real(srp) function the_neurobio::percept_energy_get_current (this)

  *Get the current value of the **energy** reserves.*
- subroutine the_neurobio::percept_energy_update_current (this, current)

  *Set and update the current **energy** perception value.*
- elemental subroutine the_neurobio::percept_energy_destroy_deallocate (this)

  *Destroy the **energy** perception object and deallocate.*
- elemental subroutine the_neurobio::percept_age_create_init (this)

  *Initiate an empty **age** perception object.*
- elemental integer function the_neurobio::percept_age_get_current (this)

  *Get the current value of the **age** reserves.*
- subroutine the_neurobio::percept_age_update_current (this, current)

  *Set and update the current **age** perception value.*
- elemental subroutine the_neurobio::percept_age_destroy_deallocate (this)

  *Destroy the **age** perception object and deallocate it.*
- subroutine the_neurobio::spatial_percept_set_cid (this, id)

  *Set unique **id** for the conspecific perception component.*
- elemental integer function the_neurobio::spatial_percept_get_cid (this)

  *Get the unique **id** of the food item object.*
- elemental subroutine the_neurobio::consp_percept_comp_create (this)

  Create a single conspecific perception component at an undefined position with default properties.
- subroutine the_neurobio::consp_percept_make (this, location, size, mass, dist, cid, is_male)

  Make a single conspecific perception component. This is a single conspecific located within the visual range of the agent.
- elemental real(srp) function the_neurobio::consp_percept_get_size (this)

  *Get the **conspecific** perception component body size.*
- elemental real(srp) function the_neurobio::consp_percept_get_mass (this)

  *Get the **conspecific** perception component body mass.*
- elemental real(srp) function the_neurobio::consp_percept_get_dist (this)

  *Get the **conspecific** perception component distance.*
- elemental logical function the_neurobio::consp_percept_sex_is_male_get (this)

  *Get the **conspecific** perception component sex flag (male).*
- elemental logical function the_neurobio::consp_percept_sex_is_female_get (this)

  *Get the **conspecific** perception component sex flag (female).*
- elemental subroutine the_neurobio::percept_consp_create_init (this, maximum_number_conspecifics)

  Create conspecifics perception object, it is an array of conspecific perception components.
- elemental subroutine the_neurobio::percept_consp_number_seen (this, number_set)

  *Set the total number of conspecifics perceived (seen) in the conspecific perception object. But do **not** reallocate the conspecific perception components so far.*
- pure subroutine the_neurobio::percept_consp_make_fill_arrays (this, consps)

  Make the conspecifics perception object, fill it with the actual arrays.
- elemental integer function the_neurobio::percept_consp_get_count_seen (this)

  Get the number (count) of conspecifics seen. Trivial.
- elemental subroutine the_neurobio::percept_consp_destroy_deallocate (this)

  Deallocate and delete a conspecific perception object.
- subroutine the_neurobio::consp_perception_get_visrange_objects (this, consp_agents, time_step_model)

Get available conspecific perception objects within the visual range of the agent, which the agent can perceive and therefore respond to.

- elemental logical function the_neurobio::consp_perception_is_seeing_conspecifics (this)

  Check if the agent sees any conspecifics within the visual range.
- elemental subroutine the_neurobio::spatialobj_percept_comp_create (this)

  Create a single arbitrary spatial object perception component at an undefined position with default properties.
- subroutine the_neurobio::spatialobj_percept_make (this, location, size, dist, cid)

  *Make a single arbitrary **spatial** object perception component.*
- elemental real(srp) function the_neurobio::spatialobj_percept_get_size (this)

  Get an arbitrary spatial object perception component size.
- elemental real(srp) function the_neurobio::spatialobj_percept_get_dist (this)

  Get the distance to an arbitrary spatial object perception component.
- real(srp) function the_neurobio::spatialobj_percept_visibility_visual_range (this, object_area, contrast, time↩ _step_model)

  Calculate the visibility range of this spatial object. Wrapper to the `visual_range` function. This function calculates the distance from which this object can be seen by a visual object (e.g. predator or prey).
- elemental subroutine the_neurobio::percept_predator_create_init (this, maximum_number_predators)

  *Create **predator** perception object, it is an array of spatial perception components.*
- elemental subroutine the_neurobio::percept_predator_number_seen (this, number_set)

  *Set the total number of **predators** perceived (seen) in the predator perception object. But do not reallocate the predator perception components so far.*
- pure subroutine the_neurobio::percept_predator_make_fill_arrays (this, preds, attack_rate)

  *Make the **predator** perception object, fill it with the actual arrays.*
- pure subroutine the_neurobio::percept_predator_set_attack_rate_vector (this, attack_rate)

  Set an array of the attack rates for the predator perception object.
- pure subroutine the_neurobio::percept_predator_set_attack_rate_scalar (this, attack_rate)

  Set an array of the attack rates for the predator perception object.
- elemental integer function the_neurobio::percept_predator_get_count_seen (this)

  Get the number (count) of predators seen. Trivial.
- elemental subroutine the_neurobio::percept_predator_destroy_deallocate (this)

  *Deallocate and delete a **predator** perception object.*
- subroutine the_neurobio::predator_perception_get_visrange_objects (this, spatl_agents, time_step_model)

  Get available predators perception objects within the visual range of the agent, which the agent can perceive and therefore respond to.
- elemental logical function the_neurobio::predator_perception_is_seeing_predators (this)

  Check if the agent sees any predators within the visual range.
- elemental subroutine the_neurobio::percept_light_create_init (this)

  Make en empty light perception component. Really necessary only when perception objects are all allocatable.
- elemental real(srp) function the_neurobio::percept_light_get_current (this)

  Get the current perception of the illumination.
- subroutine the_neurobio::percept_light_set_current (this, timestep, depth)

  *Set the current **light** level into the perception component.*
- elemental subroutine the_neurobio::percept_light_destroy_deallocate (this)

  *Destroy / deallocate **light** perception component. Really necessary only when perception objects are all allocatable.*
- subroutine the_neurobio::light_perception_get_object (this, time_step_model)

  *Get **light** perception objects into the individual PERCEPTION object layer.*
- elemental subroutine the_neurobio::percept_depth_create_init (this)

  Make en empty depth perception component. Really necessary only when perception objects are all allocatable.
- elemental real(srp) function the_neurobio::percept_depth_get_current (this)

  *Get the current perception of the **depth**.*
- subroutine the_neurobio::percept_depth_set_current (this, cdepth)

  *Set the current **depth** level into the perception component.*

- elemental subroutine the_neurobio::percept_depth_destroy_deallocate (this)

  *Destroy / deallocate **depth** perception component. Really necessary only when perception objects are all allocatable.*
- elemental subroutine the_neurobio::percept_reprfac_create_init (this)

  Make en empty reproductive factor perception component. Really necessary only when perception objects are all allocatable.
- elemental real(srp) function the_neurobio::percept_reprfac_get_current (this)

  *Get the current perception of the **reproductive factor**.*
- subroutine the_neurobio::percept_reprfac_set_current (this, reprfac)

  *Set the current **reproductive factor** level into perception component.*
- elemental subroutine the_neurobio::percept_reprfac_destroy_deallocate (this)

  *Destroy / deallocate **reproductive factor** perception component. Really necessary only when perception objects are all allocatable.*
- subroutine the_neurobio::depth_perception_get_object (this)

  *Get **depth** perception objects into the **individual** PERCEPTION object layer.*
- subroutine the_neurobio::stomach_perception_get_object (this)

  *Get the **stomach capacity** perception objects into the **individual** PERCEPTION object layer.*
- subroutine the_neurobio::bodymass_perception_get_object (this)

  *Get the **body mass** perception objects into the **individual** PERCEPTION object layer.*
- subroutine the_neurobio::energy_perception_get_object (this)

  *Get the **energy reserves** perception objects into the **individual** PERCEPTION object layer.*
- subroutine the_neurobio::age_perception_get_object (this)

  *Get the **age** perception objects into the **individual** PERCEPTION object layer.*
- subroutine the_neurobio::repfac_perception_get_object (this)

  *Get the **reproductive factor** perception objects into the **individual** PERCEPTION object layer.*
- elemental subroutine the_neurobio::percept_memory_add_to_stack (this, light, depth, food, foodsize, fooddist, consp, pred, stom, bdmass, energ, reprfac)

  Add perception components into the memory stack.
- elemental subroutine the_neurobio::percept_memory_cleanup_stack (this)

  Cleanup and destroy the perceptual memory stack.
- elemental integer function the_neurobio::percept_memory_food_get_total (this)

  Get the total number of food items within the whole perceptual memory stack.
- elemental real(srp) function the_neurobio::percept_memory_food_get_mean_n (this, last)

  *Get the **average number** of food items per single time step within the whole perceptual memory stack.*
- elemental subroutine the_neurobio::percept_memory_food_mean_n_split (this, window, split_val, older, newer)

  *Get the **average number** of food items per single time step within the perceptual memory stack, split to the first (older) and second (newer) parts. The whole memory stack ('sample') is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it.*
- elemental real(srp) function the_neurobio::percept_memory_food_get_mean_size (this, last)

  *Get the **average size** of food item per single time step within the whole perceptual memory stack.*
- elemental subroutine the_neurobio::percept_memory_food_mean_size_split (this, window, split_val, older, newer)

  *Get the **average size** of food items per single time step within the perceptual memory stack, split to the first (older) and second(newer) parts. The whole memory stack 'sample' is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it.*
- elemental real(srp) function the_neurobio::percept_memory_food_get_mean_dist (this, last, undef_ret_null)

  *Get the **average distance** to food item per single time step within the whole perceptual memory stack.*
- elemental subroutine the_neurobio::percept_memory_food_mean_dist_split (this, window, split_val, older, newer)

  *Get the **average distance** to food items per single time step within the perceptual memory stack, split to the first (older) and second(newer) parts. The whole memory stack 'sample' is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it.*
- elemental real(srp) function the_neurobio::percept_memory_consp_get_mean_n (this, last)

*Get the **average number** of conspecifics per single time step within the whole perceptual memory stack.*

- elemental integer function the_neurobio::percept_memory_predators_get_total (this)

  Get the total number of predators within the whole perceptual memory stack.

- elemental real(srp) function the_neurobio::percept_memory_predators_get_mean (this, last)

  Get the average number of predators per single time step within the whole perceptual memory stack.

- elemental subroutine the_neurobio::percept_memory_predators_mean_split (this, window, split_val, older, newer)

  *Get the **average number** of predators per single time step within the perceptual memory stack, split to the first (older) and second(newer) parts. The whole memory stack ('sample') is split by the `split_val` parameter and two means are calculated: before the `split_val` and after it.*

- elemental subroutine the_neurobio::perception_objects_add_memory_stack (this)

  *Add the various perception objects to the memory stack object. This procedure is called **after** all the perceptual components (light, depth food, conspecifics, predators, etc.) are collected (using `set` object-bound subroutines) into the perception bundle, so all the values are known and ready to be used.*

- subroutine the_neurobio::perception_objects_get_all_environmental (this)

  *A single umbrella subroutine to get all **environmental** perceptions: light, depth. This procedure invokes these calls:*

- subroutine the_neurobio::perception_objects_get_all_inner (this)

  *A single umbrella subroutine wrapper to get all **inner** perceptions: stomach, body mass, energy, age. Invokes all these procedures:*

- elemental subroutine, private the_neurobio::perception_objects_init_agent (this)

  Initialise all the perception objects for the current agent. Do not fill perception objects with the real data yet.

- elemental subroutine the_neurobio::perception_objects_destroy (this, clean_memory)

  Destroy and deallocate all perception objects.

- elemental real(srp) function the_neurobio::perception_predation_risk_objective (this)

  *Calculate the risk of **predation** as being **perceived** / **assessed** by this agent.*

- elemental real(srp) function the_neurobio::predation_risk_backend (pred_count, pred_memory_mean, weight_direct)

  Simple computational backend for the risk of predation that is used in objective risk function the_neurobio::perception_predation_risk_obje and the subjective risk function.

- elemental subroutine the_neurobio::perception_components_attention_weights_init (this, all_vals_fix, all↩ _one, weight_light, weight_depth, weight_food_dir, weight_food_mem, weight_conspec, weight_pred_dir, weight_predator, weight_stomach, weight_bodymass, weight_energy, weight_age, weight_reprfac)

  Initialise the attention components of the emotional state to their default parameter values. Attention sets weights to individual perceptual components when the overall weighted sum is calculated. The default weights are parameters defined in `COMMONDATA`.

- subroutine the_neurobio::perception_components_neuronal_response_init_set (this, this_agent, param↩ _gp_matrix_light, param_gp_matrix_depth, param_gp_matrix_food_dir, param_gp_matrix_food_mem, param_gp_matrix_conspec, param_gp_matrix_pred_dir, param_gp_matrix_predator, param_gp_matrix↩ _stomach, param_gp_matrix_bodymass, param_gp_matrix_energy, param_gp_matrix_age, param_gp↩ _matrix_reprfac, param_gerror_cv_light, param_gerror_cv_depth, param_gerror_cv_food_dir, param_↩ gerror_cv_food_mem, param_gerror_cv_conspec, param_gerror_cv_pred_dir, param_gerror_cv_predator, param_gerror_cv_stomach, param_gerror_cv_bodymass, param_gerror_cv_energy, param_gerror_cv_age, param_gerror_cv_reprfac, param_gene_label_light, param_gene_label_depth, param_gene_label_food_dir, param_gene_label_food_mem, param_gene_label_conspec, param_gene_label_pred_dir, param_gene↩ _label_predator, param_gene_label_stomach, param_gene_label_bodymass, param_gene_label_energy, param_gene_label_age, param_gene_label_reprfac)

  *Set and calculate individual perceptual components for **this** motivational state using the **neuronal response** function, for **this_agent**.*

- subroutine the_neurobio::perception_components_neuronal_response_calculate (this, this_agent, param↩ _gp_matrix_light, param_gp_matrix_depth, param_gp_matrix_food_dir, param_gp_matrix_food_mem, param_gp_matrix_conspec, param_gp_matrix_pred_dir, param_gp_matrix_predator, param_gp_matrix↩ _stomach, param_gp_matrix_bodymass, param_gp_matrix_energy, param_gp_matrix_age, param_gp↩ _matrix_reprfac, param_gerror_cv_light, param_gerror_cv_depth, param_gerror_cv_food_dir, param_↩ gerror_cv_food_mem, param_gerror_cv_conspec, param_gerror_cv_pred_dir, param_gerror_cv_predator, param_gerror_cv_stomach, param_gerror_cv_bodymass, param_gerror_cv_energy, param_gerror_cv_age,

param_gerror_cv_reprfac, perception_override_light, perception_override_depth, perception_override↩ _food_dir, perception_override_food_mem, perception_override_conspec, perception_override_pred_dir, perception_override_predator, perception_override_stomach, perception_override_bodymass, perception↩ _override_energy, perception_override_age, perception_override_reprfac)

*Calculate individual perceptual components for **this** motivational state using the **neuronal response** function, for an **this_agent**.*

- elemental real(srp) function the_neurobio::state_motivation_light_get (this)

  *Standard "get" function for the state neuronal **light** effect component.*
- elemental real(srp) function the_neurobio::state_motivation_depth_get (this)

  *Standard "get" function for the state neuronal **depth** effect component.*
- elemental real(srp) function the_neurobio::state_motivation_food_dir_get (this)

  *Standard "get" function for the state neuronal **directly seen food** effect component.*
- elemental real(srp) function the_neurobio::state_motivation_food_mem_get (this)

  *Standard "get" function for the state neuronal **food items from past memory** effect component.*
- elemental real(srp) function the_neurobio::state_motivation_conspec_get (this)

  *Standard "get" function for the state neuronal **conspecifics** effect component.*
- elemental real(srp) function the_neurobio::state_motivation_pred_dir_get (this)

  *Standard "get" function for the state neuronal **direct predation** effect component.*
- elemental real(srp) function the_neurobio::state_motivation_predator_get (this)

  *Standard "get" function for the state neuronal **predators** effect component.*
- elemental real(srp) function the_neurobio::state_motivation_stomach_get (this)

  *Standard "get" function for the state neuronal **stomach** effect component.*
- elemental real(srp) function the_neurobio::state_motivation_bodymass_get (this)

  *Standard "get" function for the state neuronal **body mass** effect component.*
- elemental real(srp) function the_neurobio::state_motivation_energy_get (this)

  *Standard "get" function for the state neuronal **energy reserves** effect component.*
- elemental real(srp) function the_neurobio::state_motivation_age_get (this)

  *Standard "get" function for the state neuronal **age** effect component.*
- elemental real(srp) function the_neurobio::state_motivation_reprfac_get (this)

  *Standard "get" function for the state neuronal **reproductive factor** effect component.*
- elemental real(srp) function the_neurobio::state_motivation_motivation_prim_get (this)

  *Standard "get" function for the root state, get the overall **primary motivation value** (before modulation).*
- elemental real(srp) function the_neurobio::state_motivation_motivation_get (this)

  *Standard "get" function for the root state, get the overall **final motivation value** (after modulation).*
- elemental logical function the_neurobio::state_motivation_is_dominant_get (this)

  Check if the root state is the dominant state in GOS.
- elemental character(len=label_length) function the_neurobio::state_motivation_fixed_label_get (this)

  Get the fixed label for this motivational state. Note that the label is fixed and cannot be changed.
- pure subroutine the_neurobio::state_motivation_attention_weights_transfer (this, copy_from)

  Transfer attention weights between two motivation state components. The main use of this subroutine would be to transfer attention from the actor agent's main motivation's attention component to the behaviour's GOS expectancy object.
- elemental real(srp) function the_neurobio::perception_component_maxval (this)

  *Calculate the **maximum** value over all the perceptual components.*
- elemental real(srp) function the_neurobio::state_motivation_percept_maxval (this)

  *Calculate the **maximum** value over all the perceptual components of this motivational state component.*
- elemental real(srp) function the_neurobio::state_motivation_calculate_prim (this, maxvalue)

  *Calculate the level of **primary motivation** for this **specific** emotional state **component**.*
- elemental subroutine the_neurobio::perception_component_motivation_init_zero (this)

  Initialise perception components for a motivation state object.
- elemental subroutine the_neurobio::state_hunger_zero (this)

*Init and cleanup **hunger** motivation object. The only difference from the base root STATE_MOTIVATION_BASE is that it sets unique label.*

- elemental subroutine the_neurobio::state_fear_defence_zero (this)

  *Init and cleanup **fear state** motivation object. The only difference from the base root STATE_MOTIVATION_BASE is that it sets unique label.*
- elemental subroutine the_neurobio::state_reproduce_zero (this)

  *Init and cleanup **reproductive** motivation object. The only difference from the base root STATE_MOTIVATION_BASE is that it sets unique label.*
- elemental subroutine the_neurobio::motivation_init_all_zero (this)

  Init the expectancy components to a zero state.
- elemental subroutine the_neurobio::motivation_reset_gos_indicators (this)

  Reset all GOS indicators for this motivation object.
- elemental real(srp) function the_neurobio::motivation_max_perception_calc (this)

  Calculate maximum value of the perception components across all motivations.
- pure real(srp) function, dimension(:), allocatable the_neurobio::motivation_return_final_as_vector (this)

  Return the vector of final motivation values for all motivational state components.
- elemental real(srp) function the_neurobio::motivation_maximum_value_motivation_finl (this)

  Calculate the maximum value of the final motivations across all motivational state components.
- elemental logical function the_neurobio::motivation_val_is_maximum_value_motivation_finl (this, test_value)

  *Checks if the test value is the maximum **final** motivation value across all motivational state components.*
- elemental logical function the_neurobio::motivation_val_is_maximum_value_motivation_finl_o (this, test_↩ motivation)

  *Checks if the test value is the maximum **final** motivation value across all motivational state components.*
- elemental subroutine the_neurobio::motivation_primary_sum_components (this, max_val)

  *Calculate the **primary motivations** from motivation-specific perception appraisal components. The **primary motivations** are motivation values before the modulation takes place.*
- elemental subroutine the_neurobio::motivation_modulation_absent (this)

  *Produce **modulation** of the primary motivations, that result in the **final motivation** values (`_finl`). In this subroutine, **modulation is absent**, so the final motivation values are equal to the primary motivations.*
- elemental subroutine, private the_neurobio::appraisal_init_zero_cleanup_all (this)

  Initialise and cleanup all appraisal object components and sub-objects.
- elemental subroutine the_neurobio::appraisal_agent_set_dead (this)

  *Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function.*
- subroutine the_neurobio::appraisal_perceptual_comps_motiv_neur_response_calculate (this)

  Get the perceptual components of all motivational states by passing perceptions via the neuronal response function.
- elemental subroutine the_neurobio::appraisal_primary_motivations_calculate (this, rescale_max_motivation)

  Calculate primary motivations from perceptual components of each motivation state.
- subroutine the_neurobio::appraisal_motivation_modulation_non_genetic (this, no_modulation)

  *Produce **modulation** of the primary motivations, that result in the **final motivation** values (`_finl`). Modulation here is non-genetic and involves a fixed transformation of the primary motivation values.*
- subroutine the_neurobio::appraisal_motivation_modulation_genetic (this, no_genetic_modulation)

  *Produce **modulation** of the primary motivations, that result in the **final motivation** values (`_finl`). Modulation involves effects of such characteristics of the agent as body mass and age on the primary motivations (hunger, active and passive avoidance and reproduction) mediated by the genome effects. Here the genome determines the coefficients that set the degree of the influence of the agent's characteristics on the motivations.*
- elemental real(srp) function asymptotic (max_level, x)

  Definition of the asymptotic function for converting the primary genotype-based modulation coefficient gamma to the actual additive multiplication value: $y = \frac{M \cdot x}{1+x}$, where M is the asymptotic value (maximum)
- elemental subroutine the_neurobio::appraisal_add_final_motivations_memory (this)

  Add individual final emotional state components into the emotional memory stack. This is a wrapper to the the_neurobio::memory_emotional::add_to_memory method.
- real(srp) function the_neurobio::reproduce_do_probability_reproduction_calc (this, weight_baseline, allow↩ _immature)

Calculate the instantaneous probability of successful reproduction.

- logical function the_neurobio::reproduction_success_stochast (this, prob)

  *Determine a stochastic outcome of **this** agent reproduction. Returns TRUE if the agent has reproduced successfully.*
- elemental subroutine the_neurobio::emotional_memory_add_to_stack (this, v_hunger, v_defence_fear, v_↩ reproduction, v_gos_label, v_gos_arousal, v_gos_repeated)

  Add emotional components into the memory stack.
- elemental subroutine the_neurobio::emotional_memory_add_gos_to_stack (this, v_gos_label, v_gos_↩ arousal, v_gos_repeated)

  Add the current GOS label or/and arousal value and/or arousal repeat count into the emotional memory stack.
- elemental subroutine the_neurobio::emotional_memory_cleanup_stack (this)

  Cleanup and destroy the emotional memory stack.
- elemental real(srp) function the_neurobio::emotional_memory_hunger_get_mean (this, last)

  Get the average value of the hunger motivation state within the whole emotional memory stack.
- elemental real(srp) function the_neurobio::emotional_memory_actve_avoid_get_mean (this, last)

  Get the average value of the fear state motivation state within the whole emotional memory stack.
- elemental real(srp) function the_neurobio::emotional_memory_reproduct_get_mean (this, last)

  Get the average value of the reproductive motivation state within the whole emotional memory stack.
- elemental real(srp) function the_neurobio::emotional_memory_arousal_mean (this, last)

  Get the average value of the GOS arousal within the whole emotional memory stack.
- subroutine the_neurobio::gos_find_global_state (this)

  *Find and set the **Global Organismic State (GOS)** of the agent based on the various available motivation values. The motivation values linked with the different stimuli compete with the current GOS and among themselves.*
- pure real(srp) function arousal_decrease_factor_fixed (time_step)
- real(srp) function arousal_decrease_factor_nonpar (time_step)
- elemental subroutine, private the_neurobio::gos_init_zero_state (this)

  Initialise GOS engine components to a zero state. The values are set to commondata::missing, commondata::unknown, string to "undefined".
- elemental subroutine the_neurobio::gos_agent_set_dead (this)

  *Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function.*
- elemental subroutine the_neurobio::gos_reset_motivations_non_dominant (this)

  Reset all motivation states as not dominant with respect to the GOS.
- elemental character(len=label_length) function the_neurobio::gos_global_get_label (this)

  Get the current global organismic state (GOS).
- elemental real(srp) function the_neurobio::gos_get_arousal_level (this)

  Get the overall level of arousal. Arousal is the current level of the dominant motivation that has brought about the current GOS at the previous time step.
- subroutine the_neurobio::gos_attention_modulate_weights (this)

  Modulate the attention weights to suppress all perceptions alternative to the current GOS. This is done using the attention modulation interpolation curve.
- elemental integer function the_neurobio::perception_food_items_below_calculate (this)

  *Calculate the number of food items in the perception object that are located **below** the actor agent.*
- elemental integer function the_neurobio::perception_food_items_below_horiz_calculate (this, hz_lower, hz↩ _upper)

  *Calculate the number of food items in the perception object that are located **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z+hz_lower, z+hz_upper].*
- elemental real(srp) function the_neurobio::perception_food_mass_below_calculate (this)

  *Calculate the average mass of a food item from all the items in the current perception object that are **below** the actor agent.*
- elemental real(srp) function the_neurobio::perception_food_mass_below_horiz_calculate (this, hz_lower, hz_upper)

*Calculate the average mass of a food item from all the items in the current perception object that are **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z+hz_lower, z+hz_upper].*

- elemental integer function the_neurobio::perception_food_items_above_calculate (this)

  *Calculate the number of food items in the perception object that are located **above** the actor agent.*
- elemental integer function the_neurobio::perception_food_items_above_horiz_calculate (this, hz_lower, hz↩_upper)

  *Calculate the number of food items in the perception object that are located **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z-hz_upper, z-hz_upper].*
- elemental real(srp) function the_neurobio::perception_food_mass_above_calculate (this)

  *Calculate the average mass of a food item from all the items in the current perception object that are **above** the actor agent.*
- elemental real(srp) function the_neurobio::perception_food_mass_above_horiz_calculate (this, hz_lower, hz_upper)

  *Calculate the average mass of a food item from all the items in the current perception object that are **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z-hz_upper, z-hz_upper].*
- elemental integer function the_neurobio::perception_conspecifics_below_calculate (this)

  *Calculate the number of conspecifics in the perception object that are located **below** the actor agent.*
- elemental integer function the_neurobio::perception_conspecifics_above_calculate (this)

  *Calculate the number of conspecifics in the perception object that are located **above** the actor agent.*
- elemental integer function the_neurobio::perception_conspecifics_below_horiz_calculate (this, hz_lower, hz_upper)

  *Calculate the number of conspecifics in the perception object that are located **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z+hz_lower, z+hz_upper].*
- elemental integer function the_neurobio::perception_conspecifics_above_horiz_calculate (this, hz_lower, hz_upper)

  *Calculate the number of conspecifics in the perception object that are located **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z-hz_upper, z-hz_upper].*
- elemental integer function the_neurobio::perception_predator_below_calculate (this)

  *Calculate the number of predators in the perception object that are located **below** the actor agent.*
- elemental integer function the_neurobio::perception_predator_above_calculate (this)

  *Calculate the number of predators in the perception object that are located **above** the actor agent.*
- elemental integer function the_neurobio::perception_predator_below_horiz_calculate (this, hz_lower, hz_↩upper)

  *Calculate the number of predators in the perception object that are located **below** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z+hz_lower, z+hz_upper].*
- elemental integer function the_neurobio::perception_predator_above_horiz_calculate (this, hz_lower, hz_↩upper)

  *Calculate the number of predators in the perception object that are located **above** the actor agent within a specific vertical horizon [hz_lower,hz_upper]. The horizon limits are relative, in that they start from the depth position of the `this` actor agent: [z-hz_upper, z-hz_upper].*
- elemental real(srp) function the_neurobio::perception_food_dist_below_calculate (this)

  *Calculate the average distance to all food items in the current perception object that are **below** the actor agent.*
- elemental real(srp) function the_neurobio::perception_food_dist_above_calculate (this)

  *Calculate the average distance to all food items in the current perception object that are **above** the actor agent.*
- elemental real(srp) function the_neurobio::perception_consp_dist_below_calculate (this)

  *Calculate the average distance to all conspecifics in the current perception object that are **below** the actor agent.*
- elemental real(srp) function the_neurobio::perception_consp_dist_above_calculate (this)

  *Calculate the average distance to all conspecifics in the current perception object that are **above** the actor agent.*
- elemental real(srp) function the_neurobio::perception_predator_dist_below_calculate (this)

*Calculate the average distance to all predators in the current perception object that are **below** the actor agent.*

- elemental real(srp) function the_neurobio::perception_predator_dist_above_calculate (this)

  *Calculate the average distance to all predators in the current perception object that are **above** the actor agent.*

- real(srp) function the_neurobio::predator_capture_probability_calculate_spatobj (this, this_predator, attack↩ _rate, is_freezing, time_step_model)

  Calculate the probability of attack and capture of the `this` agent by the predator `this_predator`. This probability is a function of the distance between the predator and the agent and is calculated by the predator-class-bound procedure the_environment::predator::risk_fish(). Example call:

- real(srp) function the_neurobio::predator_capture_probability_calculate_pred (this, this_predator, is_↩ freezing, time_step_model)

  Calculate the probability of attack and capture of the `this` agent by the predator `this_predator`. This probability is a function of the distance between the predator and the agent and is calculated by the predator-class-bound procedure the_environment::predator::risk_fish().

- real(srp) function the_neurobio::predation_capture_probability_risk_wrapper (this, is_freezing)

  Calculate the overall direct predation risk for the agent, i.e. the probability of attack and capture by the nearest predator.

- elemental real(srp) function the_neurobio::get_prop_size (this)

  Get the body size property of a polymorphic object. The object can be of the following extension of the basic the_environment::spatial class:

- elemental real(srp) function the_neurobio::get_prop_mass (this)

  Get the body mass property of a polymorphic object. The object can be of the following extension of the basic the_environment::spatial class:

Variables

- character(len= ∗), parameter, private the_neurobio::modname = "(THE_NEUROBIO)"

10.10.1 Detailed Description

The Neurobiological and behaviour architecture of the AHA Model.

Author

Sergey Budaev `sergey.budaev@uib.no`

Jarl Giske `jarl.giske@uib.no`

Date

2016-2017

10.10.2 Function/Subroutine Documentation

10.10.2.1 asymptotic()

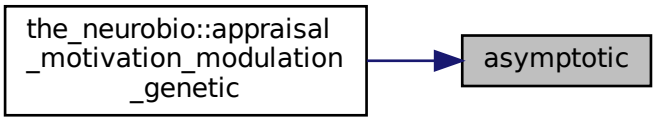
```
elemental real(srp) function appraisal_motivation_modulation_genetic::asymptotic (
            real(srp), intent(in) max_level,
            real(srp), intent(in) x )
```

Definition of the asymptotic function for converting the primary genotype-based modulation coefficient *gamma* to the actual additive multiplication value: $y = \frac{M \cdot x}{1+x}$, where M is the asymptotic value (maximum)

Note

wxMaxima quick code:
```
wxplot2d(0.7*x/(1+x), [x, 0, 10]);
```

Definition at line 6909 of file m_neuro.f90.
Here is the caller graph for this function:



10.10.2.2 arousal_decrease_factor_fixed()

```
pure real(srp) function gos_find_global_state::arousal_decrease_factor_fixed (
             integer, intent(in), optional time_step )
```

10.10.2.2.1 Fixed spontaneous arousal dissipation factor At each step, `gos_arousal` is reduced by a constant factor, `AROUSAL_GOS_DISSIPATION_FACTOR` (e.g. reduced by 0.5) independently on the current GOS time step. TODO: make dependent on the genome.

Parameters

`in`	*time_step*	time_step The number of repetitions of the same GOS.

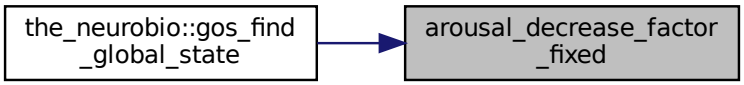
Note

Note that the `time_step` dummy parameter is **not used** in calculations here. But it is still present as an optional parameter for compatibility with the other possible dissipation pattern functions that can really depend on the time GOS repeat step.

Returns

Arousal dissipation factor.

At each GOS step the `gos_arousal` is reduced by the factor `AROUSAL_GOS_DISSIPATION_FACTOR` that is **independent** of the `time_step`.
Definition at line 7711 of file m_neuro.f90.
Here is the caller graph for this function:



10.10.2.3 arousal_decrease_factor_nonpar()

```
real(srp) function gos_find_global_state::arousal_decrease_factor_nonpar (
            integer, intent(in) time_step )
```

10.10.2.3.1 Nonparametric spontaneous arousal dissipation pattern Here the arousal dissipation factor is defined by a function of the GOS repeated time step based on polynomial or linear interpolation over a grid defined by AROUSAL_GOS_DISSIPATION_NONPAR_ABSCISSA (X) and AROUSAL_GOS_DISSIPATION_↩ NONPAR_ORDINATE (Y). For example, multiplied by 0.9 at the first time step, 0.5 at the 10s step, 0.7 at 20s step. TODO: make dependent on the genome.

Parameters

in	*time_step*	The number of repetitions of the same GOS.

Returns

Arousal dissipation factor. In this version, arousal dissipation factor is determined by a nonlinear interpolation-based function.

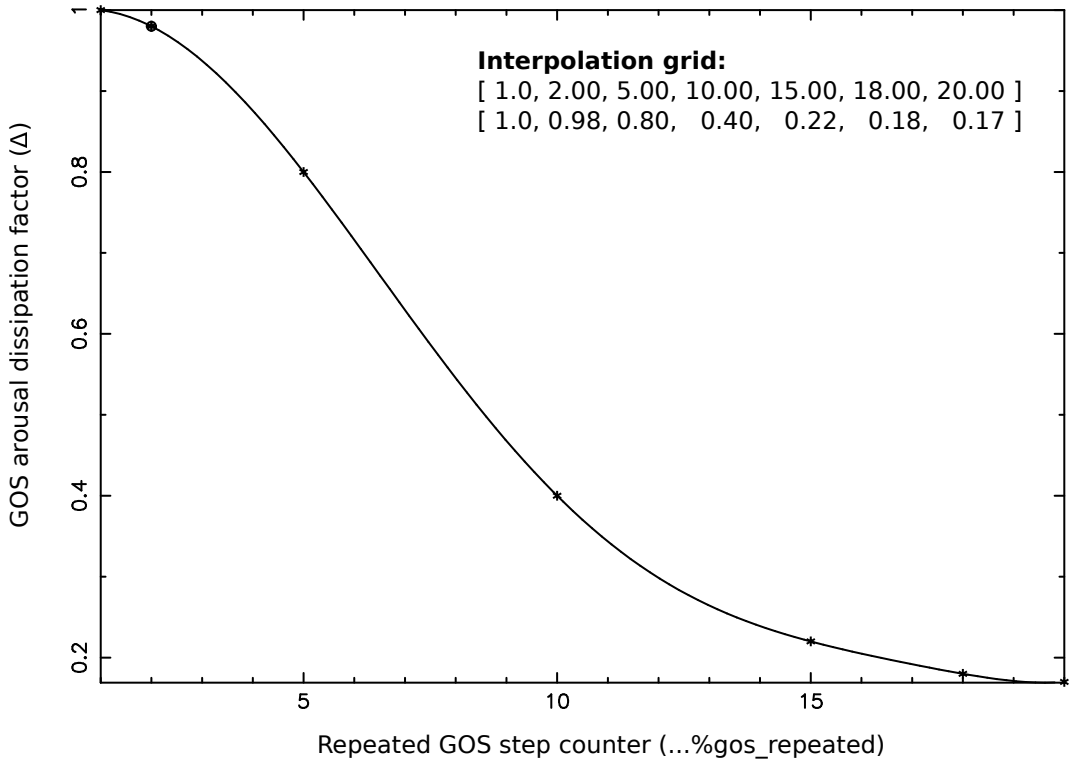


Figure 10.4 Nonparametric spontaneous arousal dissipation function

Save Interpolation plot:

```
htintrpl.exe  [ 1.0, 2.00, 5.00, 10.0, 15.0, 18.0, 20.0 ] \
              [ 1.0, 0.98, 0.80, 0.40, 0.22, 0.18, 0.17 ] [2] \
              [img_doxygen_dissipate_nonpar.ps]
```

Interpolation plots can be saved in the debug mode using this plotting command: commondata::debug_interpolate_plot_save().

Warning

Disabling the plot output allows the function to be declared as **pure** with all the benefits.

Definition at line 7753 of file m_neuro.f90.

10.11 m_popul.f90 File Reference

The Population objects for the AHA Model.

Data Types

- type the_population::member_population

  Definition of individual member of a population.
- type the_population::population

  Definition of the population object.

Modules

- module the_population

  Define the population of agents object, its properties and functions.

Functions/Subroutines

- subroutine the_population::set_individual_id (this, idnumber)

  Set integer ID number to individual member of the population object.
- integer function the_population::get_individual_id (this)

  Get integer ID number to individual member of the population object.
- subroutine the_population::individ_posit_in_environ_uniform (this, environ)

  *Places the individual agent, a member of the population, within a specific environment at random with a **uniform** distribution. The agents can be positioned with respect to their initial depth.*
- subroutine the_population::genome_individual_set_dead_non_pure (this, non_debug_log)

  *Set the individual to be **dead**. Note that this function does not deallocate the individual agent object, this may be a separate destructor function.*
- subroutine the_population::init_population_random (this, pop_size, pop_number_here, pop_name_here)

  Initialise the population object.
- subroutine the_population::population_destroy_deallocate_objects (this)

  Destroys this population and deallocates the array of individual member objects.
- subroutine the_population::population_birth_mortality_init (this, energy_mean, energy_sd)

  Impose selective mortality at birth on the agents. Selective mortality sets a fixed limit on uncontrolled evolution of the energy reserves in newborn agents. If some newborn has too high energy at birth (genetically fixed), such a deviating agent is killed at once.
- integer function the_population::population_get_popsize (this)

  Accessor get-function for the size of this population.
- integer function the_population::population_get_pop_number (this)

  Accessor get-function for the population number ID.
- character(len=label_length) function the_population::population_get_pop_name (this)

  Accessor get-function for the population character label ID.
- subroutine the_population::reset_population_id_random (this, pop_number_here, pop_name_here)

  Reset individual IDs of the population members.
- subroutine the_population::sex_initialise_from_genome (this)

  Determine the sex for each member of the population.
- subroutine the_population::position_individuals_uniform (this, environ)

  Position each member of the population randomly within a bounding environment.
- elemental subroutine the_population::sort_population_by_fitness (this)

  This subroutine sorts the population `individual` object by their %fitness components.
- recursive pure subroutine qsort (A, is_reverse)

  `qsort` is a recursive frontend for `MEMBER_POPULATION` objects
- pure subroutine qs_partition_fitness (A, marker)

partition is a pivot backend for `fitness`

- subroutine the_population::population_rwalk3d_all_agents_step (this, dist_array, cv_array, dist_all, cv_all, environment_limits, n_walks)

  Perform one or several steps of random walk by all agents.
- subroutine the_population::population_rwalk25d_all_agents_step (this, dist_array_xy, cv_array_xy, dist_↩ array_depth, cv_array_depth, dist_all_xy, cv_all_xy, dist_all_depth, cv_all_depth, environment_limits, n_↩ walks)

  Perform one or several steps of random walk by all agents.
- subroutine the_population::population_subject_predator_attack (this, this_predator, time_step_model)

  Subject the population to an attack by a specific predator. The predator acts on agents in its proximity and takes account of the predation confusion and dilution effects (see the_environment::predator::risk_fish_group()).
- subroutine the_population::population_subject_other_risks (this)

  Subject the population to mortality caused by habitat-specific mortality risk. Each agent is affected by the risk associated with the habitat it is currently in.
- subroutine the_population::population_subject_individual_risk_mortality (this)

  Subject all members of this population to their individual mortality risks.
- subroutine the_population::population_lifecycle_step_preevol (this)

  *This procedure performs a **single step** of the life cycle of the whole population, the agents for the step are selected in a random order.*
- subroutine the_population::population_lifecycle_step_eatonly_preevol (this)

  *This procedure performs a **single step** of the life cycle of the whole population, the agents for the step are selected in a random order.*
- subroutine the_population::population_save_data_all_agents_csv (this, csv_file_name, save_header, is_↩ logging, is_success)

  Save data for all agents within the population into a CSV file.
- subroutine the_population::population_save_data_all_genomes (this, csv_file_name, is_success)

  Save the genome data of all agents in this population to a CSV file.
- subroutine the_population::population_load_data_all_genomes (this, pop_size, pop_number_here, pop_↩ name_here, csv_file_name, missing_random, is_success)

  Load the genome data of all agents in this population from a CSV file. Note that the procedure implements several error correcting measures, e.g. checks for minimum number of rows in the file and minimum row length. The input CSV file therefore can include short text notes that are then ignored when reading data.
- subroutine log_write_error (message)

  This subroutine writes error message to the main log file.
- subroutine the_population::population_save_data_memory (this, csv_file_name, is_success)

  Save the perceptual and emotional memory stack data of all agents in this population to a CSV file.
- subroutine the_population::population_save_data_movements (this, csv_file_name, is_success)

  Save the latest movement history of all agents. This method makes use of the the_environment::spatial_moving::history structure that saves latest movements of each agent.
- subroutine the_population::population_save_data_behaviours (this, csv_file_name, is_success)

  Save the behaviours history stack the_neurobio::behaviour::history_behave for all agents.
- pure subroutine the_population::population_preevol_fitness_calc (this)

  *Calculate fitness for the pre-evolution phase of the genetic algorithm. **Pre-evolution** is based on selection for a simple criterion without explicit reproduction etc. The criterion for selection at this phase is set by the integer the_individual::individual_agent::fitness component. This procedure provides a whole-population wrapper for the the_individual::individual_agent::fitness_calc() function.*
- pure integer function the_population::population_ga_reproduce_max (this)

  Determine the number of parents that have fitness higher than the minimum acceptable value. Also, only alive agents are included into the reproducing number.
- real(srp) function the_population::population_ga_mutation_rate_adaptive (this, baseline, maxvalue)

  This function implements adaptive mutation rate that increases as the population size reduces.

Variables

- character(len= ∗), parameter, private the_population::modname = "(THE_POPULATION)"
- integer, public the_population::global_ind_n_eaten_by_predators

  Global indicator variable that keeps the number of agents that have died as a consequence of predatory attacks. All other dies are therefore caused by starvation.

10.11.1 Detailed Description

The Population objects for the AHA Model.

Author

Sergey Budaev sergey.budaev@uib.no
Jarl Giske jarl.giske@uib.no

Date

2016-2017

10.11.2 Function/Subroutine Documentation

10.11.2.1 qsort()

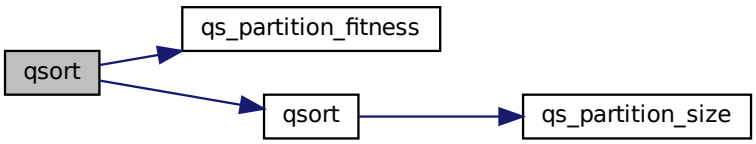
```
recursive pure subroutine sort_population_by_fitness::qsort (
            type(member_population), dimension(:), intent(inout) A,
            logical, intent(in), optional is_reverse )
```

qsort is a recursive frontend for MEMBER_POPULATION objects

Parameters

in,out	*a*	A has the same type as the individual component objects of the array-object that we are going to sort.
in	*is_reverse*	Logical flag for reverse (descending) sorting.

Definition at line 652 of file m_popul.f90.
Here is the call graph for this function:



10.11.2.2 qs_partition_fitness()

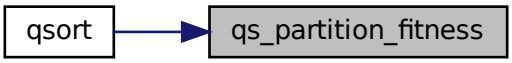
```
pure subroutine sort_population_by_fitness::qs_partition_fitness (
            type(member_population), dimension(:), intent(inout) A,
            integer, intent(out) marker )
```

partition is a pivot backend for `fitness`

Note

Pivot point `x`, has the same type **as the sorted object component**.

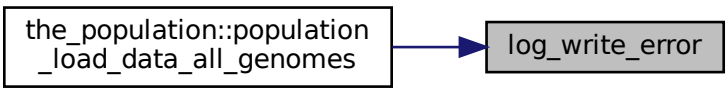
Fitness is hardwired in this partition subroutine, but it can be used as a model for similar othert sorting functions. Note that here integer array is sorted.
Definition at line 676 of file m_popul.f90.
Here is the caller graph for this function:



10.11.2.3 log_write_error()

```
subroutine population_load_data_all_genomes::log_write_error (
             character(len=*), intent(in) message )
```

This subroutine writes error message to the main log file.
Definition at line 2405 of file m_popul.f90.
Here is the caller graph for this function:



10.12 Makefile File Reference

10.13 mod_drv.f90 File Reference

The main "driver" file for the AHA Model.

Functions/Subroutines

- program aha_model_driver

  Main driver component for the AHA Model.

10.13.1 Detailed Description

The main "driver" file for the AHA Model.

Author

Sergey Budaev sergey.budaev@uib.no

Jarl Giske jarl.giske@uib.no

Date

2016-2017

10.13.2 Function/Subroutine Documentation

10.13.2.1 aha_model_driver()

`program aha_model_driver`

Main driver component for the AHA Model.

This is the actual Fortran program that calls all the computations for the AHA Model. It calls highest level procedures that are implemented in the modules:

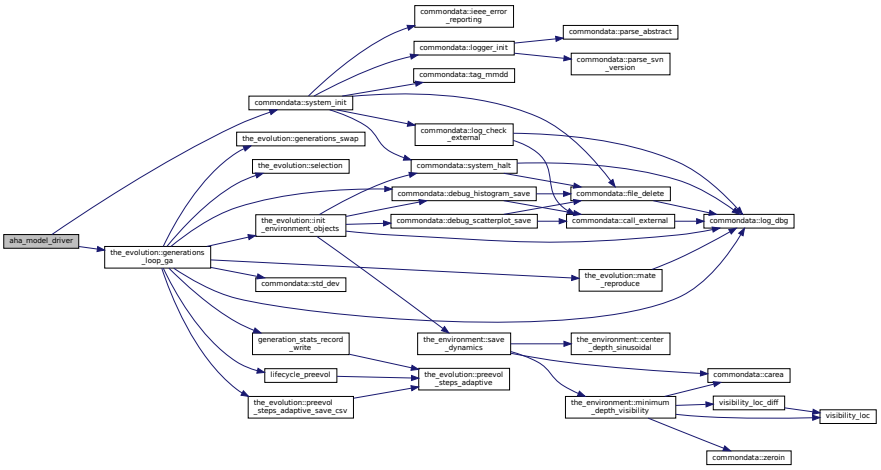
- commondata
- the_environment
- the_genome
- the_hormones
- the_body
- the_neurobio
- the_individual
- the_population
- the_evolution

File version:

```
$Id: mod_drv.f90 6145 2017-11-06 12:31:23Z sbu062 $
```

Definition at line 29 of file mod_drv.f90.

Here is the call graph for this function:



10.14 p_debug.f90 File Reference

This file contaions external procedure(s) for testing and debugging. By the separate nature, all the procedures from this file are `external`.

Functions/Subroutines

- subroutine life_cycles_debug_test

  Debugging and testing lifecycle, random wakls etc. Has been initially implemented as `DEBUG_03: block`

10.14.1 Detailed Description

This file contaions external procedure(s) for testing and debugging. By the separate nature, all the procedures from this file are `external`.

Warning

They should not be used or called from the normal model code, nor referred in the main Makefile.

Note

By the temporary and ephemeral nature of the debugging code, the coding style requirements are greatly relaxed here.

Author

Sergey Budaev `sergey.budaev@uib.no`

Date

2016-2017

10.14.2 Function/Subroutine Documentation

10.14.2.1 life_cycles_debug_test()

`subroutine life_cycles_debug_test`
Debugging and testing lifecycle, random wakls etc. Has been initially implemented as `DEBUG_03: block`

Warning

Code formatting style is not necessarily adhered to here.

PROCNAME is the procedure name for logging and debugging

10.14.2.1.1 Testing and debugging: description Testing how the procedure for joining several food resources works. Here it just joins the food resources in the safe and dangerous habitats. Because the agents are limited to stay in the safe, this just creates an extra processing overhead.
Initialise stopwatch for timing each walk.
add perceptions to the memory
Primary motivations are logged in the debug mode.
@important Save the individual data for the whole parent population, generation one, to a csv file.
Save individual data
Initialise stopwatch for timing each walk.
Update the i-th walk distance/way for distribution plot.
Introduce the cost of swimming here:
If no food objects were encountered we still grow with zero food gain.
add perceptions to the memory
Subtract the cost of living

Update increments to sex steroids.
Saving histograms of `way_passed`, the distance of each walk.
Subtract the cost of living
Save the individual data for the whole test parent population, generation one, to a csv file. This is after the walk states of the agents.
Save individual data
Definition at line 20 of file p_debug.f90.
Here is the call graph for this function:

10.15 README.md File Reference

10.16 README_NF.md File Reference