

HAFTA Overlord

Nebuloid Module API

Copyright ©2001-2003, Astra Network Inc. All Rights Reserved

January 3, 2003

This is documentation for a BETA version of HAFTA. Please be advised that you may encounter problems; please report them so that they may be fixed. The latest information about HAFTA is always available at <http://www.astranetwork.com/hafta/>.

Contents

1	Data used in the Module API	1
1.1	Types	1
1.2	Globals	2
2	Calls into a Module	2
2.1	olmod_error_t module_init();	2
2.2	olmod_blackbox_t *module_inst(pid_t process, olmod_arglist_t *static_args);	2
2.3	olmod_error_t module_set(olmod_blackbox_t *data, int64_t var);	3
2.4	int64_t module_get(olmod_blackbox_t *data, olmod_arglist_t *dynamic_args);	3
2.5	void module_deinst(olmod_blackbox_t *data);	4
2.6	void module_deinit();	4

1 Data used in the Module API

1.1 Types

olmod_error_t An enumeration of errors that originate in a module. They are defined in `include/olmod.h`.

olmod_blackbox_t A pointer to module defined data. It is the data that defines a particular instantiation of a module. The mainloop passes it

back to the module when an operation is to be done for that instance (represented by the blackbox). It is essentially a void pointer. It's contents have no meaning to anything other than the module that it originated from.

olmod_arglist_t During the `module_inst` and `module_get` calls to a module, an argument list is passed. It is essentially an array of void pointers. The mainloop will fill an argument list full of pointers to the appropriate data type(set out in the argument-type list.)

olmod_argtype_t An enumeration of types, defined in `include/olmod.h`. The module definition includes two arrays of argument types (one static and one dynamic), the mainloop will use these arrays to figure out what type of data the module wants. The list(array) is stopped with a `kEndList`.

olmod_module_t This is the module definition. It contains things like the nebuloid name, and pointers to the static and dynamic argument-type lists, as well as function pointers to the individual calls into the module.

1.2 Globals

object_global_overlord_table Every package of modules will contain a global of this name. It is a list of `olmod_module_t`'s, the overlord will look this table up and extract all nebuloid names to use in parsing the configuration file. When it needs to call into a module it will look up the functions to call in this table. This is the communication mechanism between the overlord and a module, without an entry in the table the overlord does not know a module exists.

This might be in need of a new name.

This table is created automatically by the build system.

2 Calls into a Module

Each module will define these functions. They are to be called directly from the mainloop when the mainloops finds it appropriate.

2.1 `olmod_error_t module_init();`

- Parameters: none
- Returns: an `olmod_error_t`, which is an enumeration of possible errors. `kMod_OK` is returned on "no error."
- Description: Initialization of a module. To be called soon after the `.so` has been loaded or on startup when not a `.so`. Will perform all initialization procedures that apply to the whole system (ie general data structure allocation, etc). Must be called before any other calls into this module.

2.2 `olmod_blackbox_t *module_inst(pid_t process, olmod_arglist_t *static_args);`

- Parameters:

process the pid of the process (this may have to be hacked with a bit when we are dealing with pieces of hardware instead of a process. We won't worry about exactly how to do that right now only to keep it in mind.

static_args a list containing the static arguments that are common to this module-process pair. An example would be a logging module where the static argument would be the log file, and the dynamic argument would be a dynamic argument. The data in this list is internal to `olrt`, you should never modify it directly.

- Returns: a blackbox pointer, simply a pointer to module specific data that will maintain the state of this module-process pair. Permitted to return a NULL blackbox pointer, to signify that there is no need for any state information. This is not likely because you will almost always want to store the pid in the blackbox.
- Description: Called when an module-process object is instantiated. A module must be instantiated for each process using that module. The module for a specific process must be instantiated before it may be triggered.

2.3 `olmod_error_t module_set(olmod_blackbox_t *data, int64_t var);`

- Parameters:

data the blackbox module-defined data that is associated with this module-process pair. This pointer will be kept after the instantiation call, and will be fed back to the module at each subsequent call to that module for that process.

var a 64-bit value given to the module as an input.

- Returns: an `olmod_error_t`, `kMod_OK` on success, one of the error values on failure.
- Description: Called when the action that the module performs is to be triggered with a specific value. This call has the same functionality as `module_get()` but it should be used in case where things need to be set. An example is a shared variable module that the overlord would be able to set a variable in the module.

2.4 `int64_t module_get(olmod_blackbox_t *data, olmod_arglist_t *dynamic_args);`

- Parameters:

data the blackbox module-defined data that is associated with this module-process pair. This pointer will be kept after the instantiation call, and will be fed back to the module at each subsequent call to that module for that process.

dynamic_args a list containing the dynamic arguments, ie the per trigger arguments. The data in this list is internal to `olrt`, you should never modify it directly.

- Returns: a 64-bit int, this is used in the comparisons in the mainloop.
- Description: Called when the action that the module performs is to be triggered. This is used to get a value from the module. An example is getting the value of a shared variable (that may have been set with `module_set()`) or getting the memory usage for a process.

2.5 `void module_deinst(olmod_blackbox_t *data);`

- Parameters:

 data the blackbox associated with the module-process object you would like to deinstantiate.
- Returns: nothing
- Description: Called when a module-process pair is no longer valid, as in when the process has exited. This call is essential, and should be called as soon as the process is no longer present or valid, simply because pids are reused. It also cleans up any data related to that process (ie the blackbox).
- Side Effect: blackbox data will no longer be valid after this call. ie it will be `free()`'d. Never pass it to anything again after calling this function.

2.6 `void module_deinit();`

- Parameters: none
- Returns: nothing
- Description: Cleans up any data specific to the module as a whole. It is the dual of `module_init()`. It will need to be called before a replacement module replaces the current module.