

HAFTA Release 1.0

Directory Structure

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	Specific Directories	2
1.1	/opt/bin	2
1.2	/opt/hafta/docs	2
1.3	/opt/hafta/include	2
1.4	/opt/hafta/src/checkpoint/	2
1.5	/opt/hafta/src/overlord/olrt/	2
1.6	/opt/hafta/src/overlord/olrt/modules/	2
1.7	/opt/hafta/src/overlord/ola/	2
1.8	/opt/hafta/src/overlord/olc/	2
1.9	/opt/hafta/src/overlord/mkfiles/	3
1.10	/opt/hafta/src/overlord/include/	3
2	Other Directories	3
2.1	Platform-specific directories	3
2.2	Test directories	3
3	Build environment	3
3.1	Example Structure	4

1 Specific Directories

1.1 `/opt/bin`

Binaries for the assembler (`ola`), the compiler (`olc`), and the overlord runtime (`olrt`) are included in the distribution. They are placed in `/opt/bin`, as well as links on `/usr/bin`.

1.2 `/opt/hafta/docs`

Contains HAFTA documentation in PDF format.

1.3 `/opt/hafta/include`

Contains header files for the shipped overlord nebuloid modules.

1.4 `/opt/hafta/src/checkpoint/`

The source to the checkpoint library.

1.5 `/opt/hafta/src/overlord/olrt/`

All source files that implement the runtime portion of the overlord system.

1.6 `/opt/hafta/src/overlord/olrt/modules/`

This directory should contain a subdirectory for each individual nebuloid module.

The name of the subdirectory must be the name of the module, the subdirectory name is used in the build environment to determine what modules to link into the overlord.

1.7 `/opt/hafta/src/overlord/ola/`

Overlord pcode assembler. Converts pcode assembly into binary pcode to be used directly by the runtime.

1.8 `/opt/hafta/src/overlord/olc/`

Overlord language compiler. Converts configuration language into pcode assembler to be assembled by `ola`.

1.9 /opt/hafta/src/overlord/mkfiles/

This directory contains all general makefiles to build the system. Makefiles to build individual components should reside in that components directory containing source code.

1.10 /opt/hafta/src/overlord/include/

All project-global header files will be kept here. Specific header files should be kept with the source files of the respective component.

2 Other Directories

2.1 Platform-specific directories

If there is platform-specific code it should be placed in subdirectories to separate specific code from the generic code. (eg. place all neutrino-specific code in `./nto/`, linux-specific in `./linux`)

The platform-specific directory can be subdivided even further if there is a need to separate code on an architecture-by-architecture basis.

The subdivision beyond the platform level is not yet supported

2.2 Test directories

Each point in the tree can have a test directory. This will hold any code that is required to perform a test of the functionality contained in the code at that level. Directories that contain no code can still have a test directory, but the code contained in that directory will test some general functionality. This is because there is nothing specific to test at that level.

Examples:

overlord/test/ will contain code to run tests that test the functionality of the overlord in general.

overlord/olrt/modules/test/ will contain tests that tests something general about all modules, such as the nebuloid module framework itself.

overlord/olrt/modules/mod_something/test will contain tests that only tests the functionality of the module (in this case mod_something.)

3 Build environment

Because of the fact the Overlord is a core plus a set of modules, it is very likely that end-users will want to customize what components are actually built into the system. The build system for the overlord is designed with this in mind.

The build environment is set out in such a way that there is no need for any configuration files, the makefiles are generic and they deduce what to compile by the structure of the build directory itself.

The general procedure is:

- Create a build directory (`mkdir build`)
- Copy `overlord/mkfiles/build.mk` into that directory, name it `Makefile`.
- Create directories for the components you want to build. Example: `mkdir olc`.

Eg. if there is a directory called `ola` under the build directory then it will attempt to compile a component called `ola`. So, you simply need to create the directory `ola` then run `make`. The highest-level build directory will find the makefile needed to make `ola` and place it in the `ola` directory. It will then recurse into the `ola` directory and start to make there. At this point the build is left up to the Makefile supplied by the component (typically called `component.mk`). When it finished making the build will unrecurse and look for a new directory to make (probably `olc`, or `olrt`).

So, you supply what components to build by simply making an empty directory of those component's names in the build directory.

The idea continues for those components that have sub-components (notably `olrt`). `FilenameOlrt` has the ability to compile in different modules that supply different functionality to the overlord runtime. These modules can either be compiled directly into the `filenameolrt` executable or made into a shared object package that can be dynamically linked into the `filenameolrt` executable at runtime.

So, to specify which modules are to be compiled in statically to `filenameolrt`, you simply create a directory of that modules name under `build-dir/olrt`. Eg. To create an `filenameolrt` executable with `mod_null` and `mod_var` statically linked in, you would first create the `build-dir/olrt` directory then create `build-dir/olrt/mod_null` and create `build-dir/olrt/mod_var`. When you run `make` the build environment will recurse through all the directories present, copy in the appropriate makefiles and make the `filenameolrt` executable.

To make a shared object package for use in `filenameolrt`, you simply create a directory called `build-dir/olrt/pkg_something` this will create a shared object `libpkg_something.so` in that directory. To specify which modules are to included in that package you simply make a directory for each module under that package directory, in a similar manner to specifying modules to be compiled statically.

3.1 Example Structure

To get an idea of the structure needed in the build tree, you may run 'make default' at the `overlord/` level. This will create a default build structure in the directory `build-platform`, where `platform` is your current platform. If then run `make` in that directory it will copy all the necessary makefiles into place compile the components specified.