5 HLA Internal Operation

To effectively use HLA, it helps to understand how HLA translates HLA source files into executable machine code. This information is particularly useful if you install HLA incorrectly and you cannot successfully compile a simple demo program. Beyond that, this information can also help you take advantage of more advanced HLA and OS features.

As noted earlier in the HLA documentation, HLA is not a single application; the HLA system is a collection of programs that work together to translate your HLA source files into executable files. This is not unusual, most compilers and assemblers provide only part of the conversion from source to executable (e.g., you still have to run a linker with most compilers and assemblers to produce an executable).

The HLA system offers a rich set of different configurations that allow you to mix and match components to efficiently process your assembly language applications. First of all, HLA is relatively *portable*. The compiler itself is written with Flex, Bison, C/C++, along with some platform-independent assembly language code (written in HLA, of course). This makes it easy to move the compiler from one operating system to another. Currently, HLA is supported under Windows, Linux, FreeBSD, and Mac OSX. Plans include porting HLA to NetBSD, OpenBSD, QNX and, perhaps, Solaris at some point in the future. Even within a single operating system, HLA offers multiple configurations that you can employ, based on your needs and desires. This section will describe some of the possible configurations you might create.

The compilation of a typical HLA source file using a command line such as "hla hw" goes through three or four major phases:

- The hla.exe (Windows) or hla (other OSes) program processes command-line parameters and acts as a "traffic cop" directing the execution of the remaining components of the HLA system.
- The hlaparse.exe (Windows) or hlaparse (other OSes) program is responsible for translating the HLA source file either into an object file or into the syntax of some other assembler. Usually, the hlaparse program is run automatically by some other program such as hla.exe/hla or HIDE (the HLA Integrated Development Environment). You would not normally run hlaparse directly from a command-line (though it is certainly possible to do this if you are so inclined).
- If you elect to have HLA produce an assembly language output file rather than an object module, then the next step towards producing an executable file is to run the associated back-end assembler on the source output that HLA produced. This step isn't strictly necessary because HLA can produce an object file directly without using some external assembler, but there are some (rather esoteric) reasons why you might want to go through some other assembler rather than having HLA directly produce the object file. Generally, the hla.exe (hla) program will automatically run the assembler for you.
- The last step is to run a linker to combine the object module the previous steps created with the HLA Standard Library and any other necessary object modules for the project. The output of the linkage step is an executable file (assuming, of course, there were no errors in the compilation of your program). Generally, the hla.exe (hla) program will automatically run the linker for you.

There is a fifth, optional, step that can also take place under Windows. If you are creating an application that makes use of compiled resources, as the fourth step (before the linking stage) the hla.exe (Windows only) program can run a resource compiler to translate those resources into an object module (.res) that the linker can link into your final executable.

As it turns out, the HLA system can employ a wide variety of linkers, librarians, assemblers, and other tools based on the underlying operating system. Here is the list of tools that HLA has been qualified with:

Under Windows:

- Microsoft's MASM v9 assembler
- Microsoft's linker (v9)
- Microsoft's resource compiler
- Microsofts LIB library manager
- The Flat Assembler (FASM)

- The Netwide Assembler (NASM)
- Pelles C POLINK linker
- Pelles C POLIB library manager
- Pelles C PORC resource compiler
- Borland’s Turbo Assembler

Note: you can use Borland’s TLINK and TLIB utilities with HLA, but you will have to manually run these applications; the HLA system will not automatically execute them. Note that HLA v2.0 and later have deprecated support for the Borland tools and future versions may not support them at all.

Under Linux, Mac OSX, and FreeBSD:

- The Free Software Foundation’s (FSF) Gas assembler (as)
- FSF’s linker (ld)

A couple of obvious questions that might come up: "Why provide all these options? Why not simply pick a single configuration and go with that?" Well, as it turns out, there are advantages and disadvantages to each configuration and allowing multiple configurations affords you the most flexibility when writing code.

The first point to consider is object code file versus assembly language output. The obvious choice is to have HLA produce an object code file. After all, the job of an assembler is to produce an object code file; why make the detour of producing an intermediate assembly language file that some other assembler must convert to an object file?

Historically, HLA provides this option because this was the only option available under HLA v1.x (direct object code output became available in HLA v2.0). Persons with old makefiles and other build systems may be excepting to run HLA output through some other assembler. The fact that HLA v2.x maintains the ability to produce object code in this manner preserves those existing make files and build systems.

Another reason for producing an intermediate assembly language file is because you want to see the output from the HLA compiler (for example, to see how it translates HLL-like statements into pure assembly language). The HLA compiler has an HLA output mode specifically for this purpose. It may seem silly, at first, to assemble an HLA source file into a (lower-level) HLA source file, but being able to look at the code that HLA generates is sometimes a very nice feature.

One last reason for producing an assembly language output file from an HLA source file is to allow you to translate HLA source code into the syntax for some other assembler (MASM, FASM, NASM, or Gas). This is useful when you want to combine HLA code with a project written with a different assembler.

Most of the time, of course, you’ll use the default setting and directly produce an object file from an HLA assembly. HLA produces standard PE/COFF, ELF, and Mach-o object-code files, so HLA’s output will properly link with other object files (perhaps written in different languages).