How to embed HL/C programs

Jan 14, 2023

2023 2023 Jan c hashlink haxe

HL/C is the name of the `hashlink -> C11` feature. Pulling it off is pretty easy if you follow the instructions on either the Hashlink homepage, or the Haxe manual:

```
haxe --main Main.hx --hl output/main.c
gcc -O3 -o hello -std=c11 -I out out/main.c -lhl
```

This will create a folder `out` which includes your `main.c` file, along with a ton of other auto-generated source files. These implement the classes from your program and the Haxe standard library. Here's a sample `main.c` file:

```
// Generated by HLC 4.2.5 (HL v4)
#define HLC_BOOT
#include <hlc.h>
void fun$init(void);

#include <hlc_main.c>

#ifndef HL_MAKE
#  include <hl/hashes.c>
#  include <hl/functions.c>
#  include <hl/BaseType.c>
#  include <_std/String.c>
#  include <_std/Date.c>
#  include <hl/types/ArrayAccess.c>
#  include <hl/types/ArrayBase.c>
#  include <hl/types/ArrayBytes_Int.c>
#  include <hl/types/ArrayBytes_hl_UI16.c>
#  include <hl/types/ArrayBytes_hl_F32.c>
#  include <hl/types/ArrayBytes_Float.c>
#  include <haxe/Log.c>
#  include <hl/types/ArrayDyn.c>
#  include <_std/StringBuf.c>
#  include <_std/SysError.c>
#  include <_std/Sys.c>
#  include <hl/types/ArrayObj.c>
#  include <haxe/Exception.c>
#  include <haxe/ValueException.c>
#  include <haxe/exceptions/PosException.c>
#  include <haxe/exceptions/NotImplementedException.c>
#  include <haxe/iterators/ArrayIterator.c>
#  include <haxe/iterators/ArrayKeyValueIterator.c>
#  include <hl/NativeArrayIterator_Dynamic.c>
#  include <hl/NativeArrayIterator_Int.c>
#  include <hl/types/BytesIterator_Float.c>
#  include <hl/types/BytesKeyValueIterator_Float.c>
#  include <hl/types/BytesIterator_Int.c>
#  include <hl/types/BytesKeyValueIterator_Int.c>
```

```
38 #  include <hl/types/BytesIterator_hl_F32.c>
39 #  include <hl/types/BytesKeyValueIterator_hl_F32.c>
40 #  include <hl/types/BytesIterator_hl_UI16.c>
41 #  include <hl/types/BytesKeyValueIterator_hl_UI16.c>
42 #  include <hl/types/ArrayDynIterator.c>
43 #  include <hl/types/ArrayDynKeyValueIterator.c>
44 #  include <hl/types/ArrayObjIterator.c>
45 #  include <hl/types/ArrayObjKeyValueIterator.c>
46 #  include <_std/Std.c>
47 #  include <_std/Main.c>
48 #  include <hl/_Bytes/Bytes_Impl_.c>
49 #  include <_std/Type.c>
50 #  include <haxe/NativeStackTrace.c>
51 #  include <haxe/ds/ArraySort.c>
52 #  include <hl/init.c>
53 #  include <hl/reflect.c>
54 #  include <hl/types.c>
55 #  include <hl/globals.c>
56 #endif
57
58 void hl_init_hashes();
59 void hl_init_roots();
60 void hl_init_types( hl_module_context *ctx );
61 extern void *hl_functions_ptrs[];
62 extern hl_type *hl_functions_types[];
63
64 // Entry point
65 void hl_entry_point() {
66     hl_module_context ctx;
67     hl_alloc_init(&ctx.alloc);
68     ctx.functions_ptrs = hl_functions_ptrs;
69     ctx.functions_types = hl_functions_types;
70     hl_init_types(&ctx);
71     hl_init_hashes();
72     hl_init_roots();
73     fun$init();
74 }
```

The key thing here is the `#include<hlc_main.c>` directive. This file isn't in the output folder, but rather exists in the hashlink distribution/source folder. You need to make sure the compiler can find this file, otherwise it won't work. All of the other included files will exist in the auto-generated `out` folder, so they're not a concern.

Also, we can see that there's a `HL_MAKE` define wrapping all of the includes. This will come in handy later when adding a build system, so that we can compile those files individually and have much faster build times.

What is hlc_main.c?

This file is the real entrypoint of the program. If you notice, there's no `main` function above. Instead, `hl_entry_point` gets called by hlc_main. This is how you can compile a standalone program with hashlink; hlc_main implements everything you need to create a basic program.

Unfortunately, it's not very useful if we're trying to embed hashlink into an existing program. Instead, we'll want to implement that stuff ourselves.

Embedding as a library

We can toss the compiler a dummy `hlc_main.c` file, and compile the HL/C output into a standard library that we can link into our host program. If we want to use our Haxe code as a program, we can reimplement parts of `hlc_main` so that we can call `hl_entry_point` as usual. If instead we wish to do something more complicated, like having multiple entrypoints, or directly calling/instantiating classes, then we will need to familiarize ourselves with some of hashlink's APIs.

Working with C (Index)