To create the current version, get & unpack the current distribution, bootstrap it, and do the following
in the root directory of the oo2c tree:

oo2c -M src/TestH2O.Mod

on my system this results in the following error messages (ðboring, therefore folded awayï), but it creates a runnable version of TestH2O in the directory bin.

Now, I've created an test.h2o in directory misc with the following contents:

```
H2O {
  OPTIONS {
    MapChar = "SHORTCHAR";
    MapShort = "SHORTINT";
    MapLong = "INTEGER";
    MapLongLong = "LONGINT";
    MapFloat = "SHORTREAL";
    MapDouble = "REAL";
    MapPointer = "ANYPR";
  }

  MODULE "test" {
      LinkLib = "libtest.so";
  }

  MODULE "mod" {
      LinkLib = "libmod.so";
  }
}
#include "test.c"
```

and call TestH2O from the directory one level higher:

TestH2O misc/test.h2o

getting the following result:

```
 Treutwein@FRB3320D4Y ~/H2O
 $ TestH2O.exe misc/test.h2o
 NEW MODULE test for misc/test.h2o
 AddImports
 <Processing misc/test.h2o>
MapChar = "SHORTCHAR";
MapUnsignedChar = "CHAR";
MapShort = "SHORTINT";
MapUnsignedShort = "INTEGER";
MapLong = "INTEGER";
MapUnsignedLong = "LONGINT";
MapLongLong = "LONGINT";
MapUnsignedLongLong = "HUGEINT";
MapFloat = "SHORTREAL";
MapDouble = "REAL";
MapLongDouble = "LONGDOUBLE";
MapPointer = "ANYPR";
MapEnum = "LONGINT";
MapVoid = "C_VOID";
OutputDirectory = ".";
RenameVariables = FALSE;
RenameProcedures = TRUE;
TagSuffix = "_tag";
AutoPrefix = "Auto";
ModuleSuffix = "Mod";
Include = ".";
Exclude = ;
Framework = "/System/Library";
ParserOutputDeclarations = FALSE;
ParserOutputDependencies = FALSE;
```

```
OutputSearchFile = FALSE;
OutputFrameworkPath = FALSE;
AllowRedefinedProc = FALSE;
AllowRedefined = FALSE;
StripPrefix = ;
OutputName = "test";
InterfaceType = "C";
Prolog = "";
Epilog = "";
LinkFramework = ;
LinkLib = "libtest.so";
LinkFile = ;
Import = ;
DefaultVar = FALSE;
Translate = TRUE;
Merge = FALSE;
StripPrefix = ;
OutputName = "mod";
InterfaceType = "C";
Prolog = "";
Epilog = "";
LinkFramework = ;
LinkLib = "libmod.so";
LinkFile = ;
Import = ;
DefaultVar = FALSE;
Translate = TRUE;
Merge = FALSE;
 MODULE test for misc/test.c
 AddImports
 <Processing misc/test.c>
 MODULE mod for misc/mod.h
 AddImports
 <Processing misc/mod.h>
```

```
    MODULE test IMPORTS mod
    End of line assumed for incorrectly terminated file
   At positon 364 (line=24, col=0) in file misc/test.h2o
    Processing: mod
    Processing: test
    Processed 74 lines in 3 files
```

and that looks not bad ...

```
------------------------------------------------------------
Date: Mon, 11 Dec 2006 22:34:36 +0900
From: sgreenhill@iinet.net.au
To: Treutwein Bernhard <Bernhard.Treutwein@Verwaltung.Uni-
Muenchen.DE>
Subject: Re: TestH2O - Problems
-------------
Hi Bernhard,

You have to tell H2O the name of the top level file.
Normally, this file will include a H2O section with options
and module definitions and finish with a #include directive
that includes the actual source files that you want to
translate.

So if you want to translate test.c, you would make test.h2o
that looks like this:

H2O {
  ... translation options here ...
}
#include "test.c"

To initiate the translation you would do:

 TestH2O test.h2o
```

For included modules, the behaviour depends on the type of include. If you do:
  #include <mod.h>
it looks in the search path specified by the Include option. If you do:
  #include "mod.h"
it looks in the same directory as the file that does the #include.

Off the top of my head, I'm not sure of the default values (eg. whether it searches the current directory). I don't have the details to hand, but I'll check them tomorrow and let you know.

If you like, I can send you some working examples. Are there any particular APIs that you are trying to translate?

Cheers,
  Stewart

Date: Fri, 05 Aug 2005 10:08:23 +0800
From: Stewart Greenhill <sgreenhill@iinet.net.au>
To: Treutwein Bernhard <Bernhard.Treutwein@Verwaltung.Uni-Muenchen.DE>

Hi Bernhard,

There are a couple of modes in which it can work.

With arguments "--preprocess" it just preprocesses source, outputting tokenised symbols (not very useful).

With arguments "--preprocess --text" it preprocesses source, outputting text.

Without arguments, it translates "C" definitions, producing Oberon-2 output. For example, if you unpack the first attached archive "misc.tar.gz", you should be able to do:

```
TestH2O misc/test.c
```

It should produce something like this:

```
TestH2O.exe misc/test.c
NEW MODULE test for misc/test.c
AddImports
<Processing misc/test.c>
NEW MODULE mod for misc/mod.h
AddImports
<Processing misc/mod.h>
MODULE test IMPORTS mod
Processing: mod
Processing: test
Processed 51 lines in 2 files
```

...and will output two files (one for each source file): test.Mod, and mod.Mod. Check the definitions in these files, and you'll get a feel for how the C types are translated.

Also attached are some more complex examples ("interfaces.tar.gz"). For example, the opengl interface is translated as a set of modules (look under "src" directory for the generated files). There is a configuration file "opengl.h2o" which contains various directives about how the translation is to be done. On the other hand, "opencv.h2o" merges multiple include files into compound modules (using the "Merge" directive). Take a look at the files, and you should get a feel for how it works.

There is an outer "H2O" directive. Otherwise, all text is

processed as "C" source code.

The "OPTIONS" directive is global to the translation, and includes things like:

- OutputDirectory - where to put the generated module files
- Include - list of paths to get include files
- Exclude - list of files to not include (#include ignored)
- AutoPrefix - prefix for auto-generated type names (default "Auto")
- TagSuffix - suffix to be added to structure tags (default "_tag")
- RenameProcedures - when set to "1" (the default) renames procedures using module "StripPrefix" specification.
- RenameVariables - when set to "1" (default is "0", since this is unsupported in OOC V2) renames variables using module "StripPrefix" specification.
- ModuleSuffix - suffix for module file names (default "Mod").

In this section, you can also specify the type mappings for scalar types:

OPTION symbol	C type	Default output type
MapChar	"char"	"CHAR"
MapUnsignedChar	"unsigned char"	"CHAR"
MapShort	"short"	"INTEGER"
MapUnsignedShort	"unsigned short"	"INTEGER"
MapLong	"long"	"LONGINT"

MapUnsignedLong	"unsigned long"	"LONGINT"
MapLongLong	"long long"	"HUGEINT"
MapUnsignedLongLong	"unsigned long long"	"HUGEINT"
MapFloat	"float"	"REAL"
MapDouble	"double"	"LONGREAL"
MapLongDouble	"long double"	"LONGDOUBLE"
MapPointer	"void *"	"SYSTEM.PTR"
MapEnum	"enum ..."	"LONGINT"
MapVoid	"void"	"C_VOID"

Each "MODULE" directive controls how to treat the named module. The base module name comes from the name of the corresponding ".h" file. Options per module include:

OutputName - Output name for this module (defaults to header name)

StripPrefix - list of prefixes to be removed from symbol names

LinkLib, LinkFile, LinkFramework - specifies libraries, files and frameworks to link to this module (OOC-specific directive)

Prolog - "C" definitions to be processed at the start of this module

Epilog - "C" definitions to be processed at the end of this module

Merge - when set to "1", causes all files included by this module to be declared within this module, rather than in separate modules.

Each "VARIANT" directive specifies how to translate particular symbols. As you would know, there are many ways of interpreting C declarations, and the default rules don't always work. For example:

```
f(char * arg);
```

could be:

```
PROCEDURE f (arg : POINTER TO ARRAY OF CHAR);
PROCEDURE f (arg : ARRAY OF CHAR);
PROCEDURE f (VAR arg : ARRAY OF CHAR);
PROCEDURE f (VAR arg : CHAR);
```

So VARIANT directives allow you to place particular interpretations on the "C" declarations. Normally, you can see some specific patterns, but it varies from API to API.

The format of variants is a form of designator. The designators are composed of:

strings -	which match the names of globally declared objects
[n] -	which matches item <n> in a compound type or parameter list
^ -	which matches the item referenced by a pointer type
.symbol -	which matches a named item in a type or paramter list

Strings or symbols use OOC's regexp string format. Basically, ".*" matches an string of characters, "$" matches the end of a string, and "[]" matches one of a set of characters.

Examples:

```
"gl.*v$".params : ARRAY;
```

This means any symbol (here, procedure) starting with "gl" and ending with "v" has its "params" parameter interpreted as an array.

```
"gl.*Matrix[fd]$"[0] : ARRAY;
```

This means any symbol starting with "gl" and ending with

"Matrixf" or "Matrixd" has its first parameter interpreted as an array.

"String$" : CSTRING POINTER;
This means that the type "String" is to be interpreted as a POINTER to a C string (in OOC, it assigns the "CSTRING" attribute to the pointer, which means that you can use a string literal or CHAR array for this type).

"glutInit$".argcp : VAR;
This means that the "argcp" parameter of function "glutInit" is to be treated as a VAR parameter.

"cvRelease[^D].*$"[0] : VAR;
This means that symbols starting "cvRelease" followed by any character EXCEPT "D" has its first parameter interpreted as a VAR parameter.

Hope this helps you get started. The new H2O is much more flexible, but not as well tested as the old version. If you hit a snag, I am happy to try to help you out. Error reporting is patchy in some places, so it might not be clear exactly what is happening if something goes wrong.

Cheers,
Stewart