How to compile FEFLOW IFM plug-ins in Linux v0.2

Axayacatl Maqueda
axa.maqueda@protonmail.ch

October 11, 2017

1 Objective

Compile FEFLOW IFM plug-in as dynamic libraries in Linux environment.

Note 1: Some steps may be redundant or not necessary to accomplish the objective of compiling IFM plug-ins because the process was trial & error. However, I dont know which are the redundant steps.

Note 2: compilation under Ubuntu 14.04 Desktop and FEFLOW 7. Root access is needed to compile IFM plug-ins.

Note 3: last test on February 2016.

2 FEFLOW install

Installs the FEFLOW components, while installing the deb packages theres a warning message telling the proper usage dpkg command and flags

1. `wasy-core_5.1-4_amd64.deb`
2. `wasy-feflow_6.2-8_amd_64.deb`
3. `wasy-qt_4.8-6_amd_64.deb`
4. `wasy-feflow-ifmdevel_6.28_all.deb`
5. `wasy-feflow-benchmarks_6.2-0_all.deb`

3 Install missing commands and libraries

Plug-ins are compiled using the `imake` command. To test if this command is installed type `xmkmf` in a terminal window. The following message should appear:

```
imake DuseInstalled I/usr/lib/X11/config
```

If the command is not found install X Window System utilities with:

```
apt-get install xutils-dev
```

If during compilation a header or a library is missing the package that contains it can be tracked by apt-file search command. If not installed:

```
apt-get install apt-file
apt-file update
```

The missing library or header can be located by:

```
apt-file search <missing header name>
```

The Ubuntu 14.04 installation also missed `libxt-dev` and `libmotif-dev` packages.

UPDATE `GCC`, `G++` and `GNU MAKE` to recent versions

GCC 5.3.0 (released 2015) (22 Feb 2016) G++ 5.3.0 (released 2015) (22 Feb 2016)

`gcc 5` on Ubuntu 14.04 https://gist.github.com/beci/2a2091f282042ed20cda

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt-get update
sudo apt-get install gcc-5 g++-5
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-5 60 --slave /usr/bin/g+
```

GNU MAKE 4.1 (released 2014) (installed 22 Feb 2016)

how to update make 3.81 linux http://stackoverflow.com/questions/31912233/how-to-update-make-3

```
cd /tmp
wget http://ftp.gnu.org/gnu/make/make-4.1.tar.gz
tar xvf make-4.1.tar.gz
./configure
make
sudo make install
rm -rf make-4.1.tar.gz make-4.1
```

make it (make 4.1) the default make by prefixing `/usr/local/bin` to your `\$PATH` variable in your shell startup file

```
PATH=\$PATH:/usr/local/bin
export PATH
```

4 Compiling IFM source files (optional?)

Found this information of FEFLOW users forum, not sure if its right but applied it anyways:

Compiling IFM module under Linux http://forum.mikepoweredbydhi.com/index.php/topic,1955.msg4394.html#msg4394

Step 1 for FEFLOW 6.2: compile `libifm` from SDK (this can be done in `/opt/wasy/sdk/ifm/src`)

```
cd /opt/wasy/sdk/ifm/src
gcc fPIC I/opt/wasy/sdk/ifm/include I/opt/wasy/sdk/ifm/include/xdk c *.c
ar rcs /opt/wasy/sdk/ifm/lib/libifm.a *.o
```

Step 1 for FEFLOW 7.0 (installation path changes to `/opt/feflow/`

```
cd /opt/feflow/sdk/ifm/src
gcc fPIC I/opt/feflow/sdk/ifm/include I/opt/feflow/sdk/ifm/include/xdk c *.c
ar rcs /opt/feflow/sdk/ifm/lib/libifm.a *.o
```

Using step 2: works well for `HelloWorld.cpp` example described in Creating an Example IFM Module with FEFLOW 5.4 and MS Visual Studio 2008. However, step 2 does not work for more complex plug-ins.

5 Compile an example IFM plug-in

Sample IFM codes were installed in directory `/opt/wasy/sdk/ifm/samples` for FEFLOW6 and on directory `/opt/feflow/sdk/ifm/samples/` for FEFLOW7. Copy the directory all the files in `/sdk/ifm/samples/simul/ifm\_prop/` to a directory in `/home/user` to ensure write access, i.e.`/home/user/plugin`. The files in the directory are the following:

1. `ifm_prop.cpp`
2. `ifm_prop.h`
3. `ifm_prop.htm`
4. `ifm_prop.txt`
5. `Imakefile`

5.1 Editing the Imakefile

The first lines of Imakefile are only comments,

```
#ifndef XCOMM
#define XCOMM #
#endif
XCOMM
XCOMM************************ Copyright (C) WASY Ltd. 2003 ************************
XCOMM
XCOMM       FEFLOW * interactive graphics-based Finite Element simulation
XCOMM            system for subsurface FLOW and transport processes
XCOMM
XCOMM*******************************************************************************
XCOMM
```

Figure 1: first lines in Imakefile

5.2 Add the plug-in to the simulation

The following section defines the language code and the location of the IFM template, it is necessary to leave it as `#include </opt/feflow/sdk/ifm/cf/Ifm.tmpl>` or the appropriate location,

```
#define CplusplusSource
#undef  CCsuf
#define CCsuf cxx

#include <../../../cf/Ifm.tmpl>
```

Figure 2: .

In next section of the `Imakefile` its necessary to modify the value of IFMDIR and leave it as `IFMDIR = /opt/feflow/sdk/ifm` (without the / after ifm)

```
/*************************************************************************/

          XPM_LIBS = # we do not need XPM ...
            IFMDIR = ../../..
```

Figure 3: .

The following section was created by FEFlow 5.x (apparently). It contains:

```
XCOMM -- IFM_BEGIN (Begin of IFM maintained section) --------------

           THISFILE = __FILE__

            MODNAME = SAMPLE_CXX

            PRIMSRC = ifm_prop.cpp
            DSONAME = ifm_prop.so
             DSOREV = 1

            REGPROC = RegisterModule
           HTMLFILE = ifm_prop.htm
          COPYRIGHT = ifm_prop.txt

            GUILIBS =

      IFMDEBUGFLAGS = DebuggableCDebugFlags -fPIC
         IFMLDFLAGS = -Wl,--no-undefined
        IFMINCLUDES = -I. $(IFMINC)
         IFMDEFINES =
         IFMDEPLIBS =  $(DEPIFMLIB)
            IFMLIBS =  $(IFMLIB)

            IFMOBJS = ifm_prop.o
            IFMSRCS = $(PRIMSRC)

XCOMM -- IFM_END (End of IFM maintained section) ------------------
```

Figure 4: .

module name (MODNAME), this name should be equal to the one declared in the cpp file in the following line: `IfmRegisterModule(pMod, SIMULATION, IFM_PROP, IFM Properties, 0x1000);` The name should equal to the second value in pink = `IFM_PROP` see fig 5

```
IfmResult RegisterModule(IfmModule pMod)
{
  if (IfmGetFeflowVersion (pMod) < IFM_REQUIRED_VERSION)
    return False;
  g_pMod = pMod;
  IfmRegisterModule (pMod, "SIMULATION", "IFM_PROP", "IFM Properties", 0x1000);
  IfmSetDescriptionString (pMod, szDesc);
```

Figure 5: .

PRIMSRC: this value should be equal to the file containing the C++ code
DSONAME: this value is the desired name for the dynamic library
HTMLFILE: plug-in help file
COPYRIGHT: txt copyright file
GUILIBS: this value should be `lXm lXt lX11` to make it compatible with FEFLow 6.x
IFMDEBUGFLAGS: make sure `fPIC` is there to ensure the plug-in is Position Independent Code
IFMOBJS: this is an intermediate file in the way to the final .so library.

The final part of the Imakefile needs **ONE** very important modification, see fig 6. Add to `CXXDEBUGFLAGS` the following flag `std=c++11`

This flag indicates that the g++ compiler must use the c++ 11 standard. https://gcc.gnu.org/projects/cxx0x.html The use of this flag ensures the code written and debugged on Visual Studio 2013 on Windows can be compiled on Linux. More details on Appendix 2

```
XCOMM -- IFM_END (End of IFM maintained section) ------------------

      CXXDEBUGFLAGS = $(IFMDEBUGFLAGS)
        CDEBUGFLAGS = $(IFMDEBUGFLAGS)
           INCLUDES = $(IFMINCLUDES)
            DEFINES = $(IFMDEFINES) -DIFM_NO_X11
            DEPLIBS = $(IFMDEPLIBS)
               LIBS = $(IFMLIBS) $(GUILIBS)

               OBJS = $(IFMOBJS)
               SRCS = $(IFMSRCS)

all:: $(DSONAME)

/*************************************************************************/

IfmModuleTarget($(DSONAME),$(DSOREV),$(OBJS),$(DEPLIBS),$(LIBS),$(SYSLIBS))
DependTarget()
```

Figure 6: .

The final version of `Imakefile` for `ifm_prop` is found in Appendix 1 ready to copy it into a new file.

5.3 Compilation

On `/home/user/plugin/ifm_prop` type `xmkmf`. A file named Makefile is created in the same directory. Now type `make -d`. The `-d` flag is display verbose debug messages. Theres the following fatal error, fig 7

```
ifm_prop.cpp:1:20: fatal error: stdifm.h: No such file or directory
 #include "stdifm.h"
```

Figure 7: .

The header `stdifm.h` is not located on the directory. This header is not found on the SDK Ubuntu install, import it from any functional IFM project in Windows.

Once copied into `/home/user/plugin/ifm_prop` run again `xmkmf` and `make -d` If the compilation was successful the following files were created on the directory:

1. Makefile: created by `xmkmf`, even if the Imakefile is wrong dont expect error messages here.
2. `ifm_prop.o` : intermediate file created by xmkmf)
3. `ifm_prop.so.1`
4. `ifm_prop.so`: compiled plug-in to be attached to FEFLOW GUI)

5.4 Plug-in test

Run `feflow70q` and add for `ifm_prop.so` from the Available Plug-ins panel. Open any FEM file, attach the plug-in, right click & edit, the result is the following fig 8:

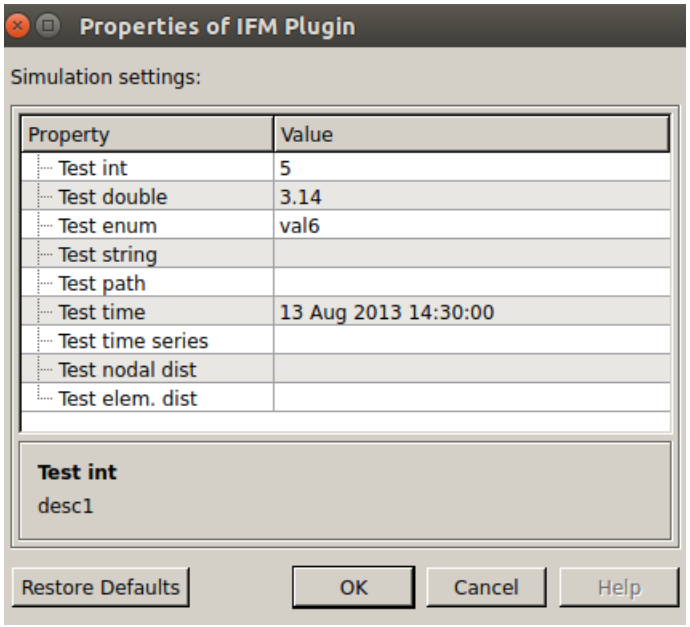


Figure 8: .

6 Import a plug-in developed in MS Visual Studio 2013

Visual Studio Projects are generated by default in `C:/users/user/Documents/Visual Studio 2013/Proje`

For example a successfully debugged project named `export_data_3` looks like this, fig 9:

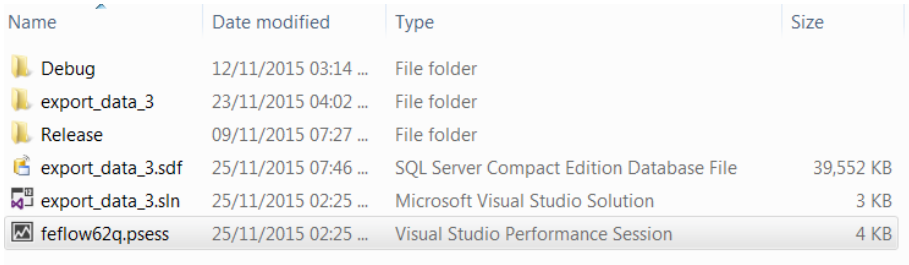
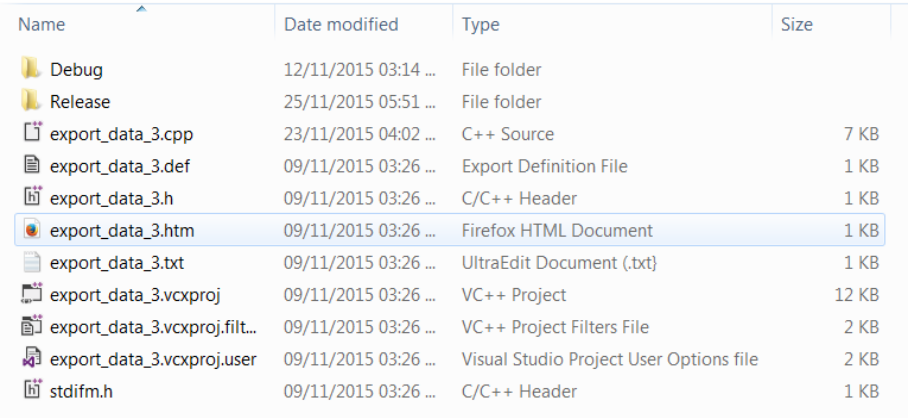


Figure 9: .

The important files are located on the `export_data_3` subfolder, inside it looks like this, fig 10:



Name	Date modified	Type	Size
Debug	12/11/2015 03:14 ...	File folder	
Release	25/11/2015 05:51 ...	File folder	
export_data_3.cpp	23/11/2015 04:02 ...	C++ Source	7 KB
export_data_3.def	09/11/2015 03:26 ...	Export Definition File	1 KB
export_data_3.h	09/11/2015 03:26 ...	C/C++ Header	1 KB
export_data_3.htm	09/11/2015 03:26 ...	Firefox HTML Document	1 KB
export_data_3.txt	09/11/2015 03:26 ...	UltraEdit Document (.txt)	1 KB
export_data_3.vcxproj	09/11/2015 03:26 ...	VC++ Project	12 KB
export_data_3.vcxproj.filt...	09/11/2015 03:26 ...	VC++ Project Filters File	2 KB
export_data_3.vcxproj.user	09/11/2015 03:26 ...	Visual Studio Project User Options file	2 KB
stdifm.h	09/11/2015 03:26 ...	C/C++ Header	1 KB

Figure 10: .

Actually, only 3 files are needed:

1. `export_data_3.cpp`
2. `export_data_3.h`
3. `stdifm.h`

Import them into a directory in the Ubuntu install and then:

Copy an Imakefile into the same directory
Edit the Imakefile as explained in section 4.1
Run `xmkmf` and `make -d`
Attach plug-in to FEM file.

7 Code compiles/works on VS 2013 but not on Linux g++

`pow` function needs the header math.h to be declared on the code, thus add `#include <math.h>` to plug-in code

function causes the following error during compilation `error: crosses initialization of <variable>`
The following code will produce the error:

Listing 1: .n

```
(x<100) goto no_changes;          // goto somewhere in code
double sum = 1 + x;               // variable declaration + calculation
no_changes:                                 // goto reference
```

change code to the following:

Listing 2: .

```
if (x<100) goto no_changes;       // goto somewhere in code
double sum;                       // variable declaration
sum = 1 + x;                      // calculation
no_changes:                       // goto reference
```

also add the flag `fpermissive` to the following lines in the Imakefile, fig 11

```
50 XCOMM -- IFM_END (End of IFM maintained section) ------------------
51
52       CXXDEBUGFLAGS = $(IFMDEBUGFLAGS) -fpermissive
53         CDEBUGFLAGS = $(IFMDEBUGFLAGS) -fpermissive
```

Figure 11: .

6.3 c++ 11 support (see 4.1 editing the Imakefile and Appendix 2)

6.4 time.h functions that work on VS2013 do not compile under G++ 5.3.0 . Functions like `__time64_t, localtime_s, asctime_s`. Instead `time_t, localtime, asctime` functions do work.

6.5 Backslash \ & forward slash /. Windows uses backslash for paths, Linux uses forward slashes for paths. If in the IFM code functions such as `IfmGetProblemPath` and then the variable value is concatenated to generate new values, take into account this issue.

Windows: "import+export\\"
Linux: "import+export/"

7.1 7 Using the plug-in in another system different where it was compiled

Plug-ins are compiled as .so or dynamic libraries. So, they only can run on the system where they were compiled. If the location of `/wasy/` install directory is the same, compile the Imakefile with `xmkmf` and then `make -d`. If not, edit the Imakefile according to the location of `/wasy/sdk/` directory

8 Appendix 1

Listing 3: .

```
#ifndef XCOMM
#define XCOMM #
#endif
XCOMM
XCOMM************************ Copyright (C) WASY Ltd. 2003 ***************
XCOMM
XCOMM       FEFLOW * interactive graphics-based Finite Element simulation
XCOMM            system for subsurface FLOW and transport processes
XCOMM
XCOMM*****************************************************************
XCOMM

#define CplusplusSource
#undef  CCsuf
#define CCsuf cxx

#include </opt/wasy/sdk/ifm/cf/Ifm.tmpl>

/*****************************************************************/

           XPM_LIBS = # we do not need XPM ...
             IFMDIR =  /opt/wasy/sdk/ifm

XCOMM -- IFM_BEGIN (Begin of IFM maintained section) --------------

           THISFILE = __FILE__

            MODNAME = SAMPLE_CXX

            PRIMSRC = ifm_prop.cpp
            DSONAME = ifm_prop.so
             DSOREV = 1

            REGPROC = RegisterModule
           HTMLFILE = ifm_prop.htm
          COPYRIGHT = ifm_prop.txt

            GUILIBS =  -lXm -lXt -lX11

      IFMDEBUGFLAGS = DebuggableCDebugFlags -fPIC
         IFMLDFLAGS = -Wl,--no-undefined
        IFMINCLUDES = -I. $(IFMINC)
         IFMDEFINES =
         IFMDEPLIBS =  $(DEPIFMLIB)
            IFMLIBS =  $(IFMLIB)
```

```
            IFMOBJS = ifm_prop.o
            IFMSRCS = $(PRIMSRC)

XCOMM -- IFM_END (End of IFM maintained section) ------------------

      CXXDEBUGFLAGS = $(IFMDEBUGFLAGS)   s t d =c++11
        CDEBUGFLAGS = $(IFMDEBUGFLAGS)
           INCLUDES = $(IFMINCLUDES)
            DEFINES = $(IFMDEFINES) -DIFM_NO_X11
            DEPLIBS = $(IFMDEPLIBS)
               LIBS = $(IFMLIBS) $(GUILIBS)

               OBJS = $(IFMOBJS)
               SRCS = $(IFMSRCS)

all:: $(DSONAME)

/*************************************************************************/

IfmModuleTarget($(DSONAME),$(DSOREV),$(OBJS),$(DEPLIBS),$(LIBS),$(SYSLIBS)
DependTarget()
```

9 Appendix 2

What happens if the flag `-std=c++11` is missing

IFM code debugged in VS 2013 in Windows fails while compilation in Linux. Error messages of `make -d` do not point at the real cause of the problem. The problem is to instruct the GNU make compiler to interpret the code using the `c++11` standard. An example of a long list of error codes:

```
antamina_3.cpp:149:33: error: 'asctime_s' was not declared in this scope
  asctime_s(timebuf, 26, &newtime);    // Convert to an ASCII representation.
                                 ^
antamina_3.cpp:174:32: error: no matching function for call to 'std::basic_ifstrea
c_ifstream(std::string&)'
  ifstream inputfile(BC_filename);
                                ^
In file included from antamina_3.cpp:3:0:
/usr/include/c++/5/fstream:495:7: note: candidate: std::basic_ifstream<_CharT, _Tra
ifstream(const char*, std::ios_base::openmode) [with _CharT = char; _Traits = std::
char>; std::ios_base::openmode = std::_Ios_Openmode]
       basic_ifstream(const char* __s, ios_base::openmode __mode = ios_base::in)
       ^
/usr/include/c++/5/fstream:495:7: note:   no known conversion for argument 1 from
{aka std::basic_string<char>}' to 'const char*'
/usr/include/c++/5/fstream:481:7: note: candidate: std::basic_ifstream<_CharT, _Tra
ifstream() [with _CharT = char; _Traits = std::char_traits<char>]
       basic_ifstream() : __istream_type(), _M_filebuf()
       ^
```

Figure 12: .

All the errors get solved by adding the flag `std=c++11` to the Imakefile

```
50 XCOMM -- IFM_END (End of IFM maintained section) ------------------
51
52       CXXDEBUGFLAGS = $(IFMDEBUGFLAGS) -std=c++11
53          CDEBUGFLAGS = $(IFMDEBUGFLAGS)
54             INCLUDES = $(IFMINCLUDES)
55              DEFINES = $(IFMDEFINES) -DIFM_NO_X11
56              DEPLIBS = $(IFMDEPLIBS)
57                 LIBS = $(IFMLIBS) $(GUILIBS)
58
```

Figure 13: .