ecflow v4

ecflow v4 is no longer actively developed, only critical issues will be fixed. Please migrate to ecflow 5 at your earliest convenience

Dependencies

- cmake (install cmake (sudo apt-get install cmake))
- g++ (install g++ (sudo apt-get install g++))
- Python 2.7 or Python 3
  If you intend to use *ecFlow* Python API, You will need to install Python. (install **python-dev** (sudo apt-get install **python-dev**))
  Please ensure that *python* is accessible on $PATH otherwise, you may need to customise *$BOOST_ROOT/tools/build/v2/site-config.jam* .
  The python installation **should** include the **development** packages
  If you do not need the python api, then you can build without it, see below.
- Xlib, X11, XMotif for *ecflowview* .

  Do *not* use Lesstif library to compile ecflowview as a replacement for Motif.

  OpenMotif can be downloaded from http://www.ist.co.uk/downloads/motif_download.html
  If you do not want use the GUI, then you can configure the build to ignore this dependency.
- Qt for ecFlowUI (Qt5 preferred).
  For self-installed Qt libraries, consider setting CMAKE_PREFIX_PATH (see below). See also http://doc.qt.io/qt-5/cmake-manual.html for further details.

Setting up the build environment

- ecFlow consists of two tar files i.e.:
  - boost_1_53_0.tar.gz
  - ecFlow-4.17.0-Source.tar.gz

    Create a directory for the build:

    ```
    mkdir /tmp/ecflow_build
    ```

- Copy the two tar file into this directory, then change directory to **/tmp/ecflow_build**
- Un-zip then un-tar the two file files:

  ```
  tar -zxf boost_1_53_0.tar.gz
  tar -zxf ecFlow-4.17.0-Source.tar.gz
  ```

- You should have two directories created:

  ```
  boost_1_53_0
  ecFlow-4.17.0-Source
  ```

- Create two environment variables. These are used by some of the scripts:

  ```
  export WK=/tmp/ecflow_build/ecFlow-4.17.0-Source
  export BOOST_ROOT=/tmp/ecflow_build/boost_1_53_0
  ```

- If you have a module system, please ensure that before you start, gcc,cmake,python2,python3,etc are available in $PATH.

  ```
  module load gnu
  module load cmake
  module load python3
  module load qt
  ```

Build boost

- Boost uses bjam for building the boost libs.
  bjam source is available in boost, hence we first need to build bjam itself:

```
cd $BOOST_ROOT
./bootstrap.sh
```

- For python3

```
./bootstrap.sh --with-python=/path/to/python3
```

```
You may need to update $BOOST_ROOT/project-config.jam, with path to executable and path to include files.
```

```
# using python
#     :  # version
#     :  # cmd-or-prefix
#     :  # includes
#     :  # libraries
#     :  # condition
#     ;
using python : 3.6 : /usr/local/apps/python3/3.6.8-01/bin/python3 : /usr/local/apps/python3/3.6.8-01
/include/python3.6m ;  #  remember to preserve the spaces, as they are significant
```

- **IF** you do not require the ecFlow python API, you can avoid building boost python libs by setting.

Disable boost python, IF ecflow python API not required

```
export ECF_NO_PYTHON=1
```

before calling $WK/build_scripts/boost_build.sh (see below)
You will also need to disable python when building ecFlow. See the instruction under cmake

- ecFlow uses some of the compiled libraries in boost. The following script will build the required lib's and configure boost build according to your platform

Build boost libraries including python3 used by ecflow.

```
cd $BOOST_ROOT
$WK/build_scripts/boost_1_53_fix.sh    # fix for boost, only for some platforms
$WK/build_scripts/boost_build.sh       # compile boost libs used by ecFlow. Please see notes in
boost_build.sh, if you want to build both for python2 and python3
```

- If you want to build python2 and python3. Then ALWAYS build the python3 first. See earlier steps

Building boost python2 libs

```
module load python
mv $BOOST_ROOT/project-config.jam $BOOST_ROOT/project-config.jam_python3   # move the python3 config to
the side
./bootstrap.sh                                                             # || ./bootstrap.sh --with-
python=/path/to/python2 to regenerate project-config.jam
./b2 --with-python --clean                                                 # Clean previous python3
build *VERY* important
$WK/build_scripts/boost_build.sh                                           # Build boost python2 libs
```

Build

cmake

As configure, CMake will run some tests on the customer's system to find out if required third-party software libraries are available and note their locations (paths). Based on this information it will produce the Makefiles needed to compile and install ecFlow

CMake is a cross-platform free software program for managing the build process of software using a compiler-independent method.

Generating the Makefiles with CMake

After changing into the build ecflow directory, the user has to run CMake with his/her own options. The command gives feedback on what requirements are fulfilled and what software is still required. The table below gives an overview of the different options of configure. The default (without any options) will install in /usr/local/.

cmake options	doc	default
CMAKE_INSTALL_PREFIX	where you want to install your ecFlow	/usr/local
CMAKE_BUILD_TYPE	to select the type of compilation: • Debug • RelWithDebInfo • Release (fully optimised compiler options) • Production	Release
CMAKE_CXX_FLAGS	more flags for the C++ compiler	
ENABLE_SERVER	build the ecFlow server	on
ENABLE_PYTHON	enable python interface	on
PYTHON_EXECUTABLE	Pyhon3. Path to python3 executable. **ONLY** required if cmake version is less than 3.12.0	
ENABLE_UI	enable build of ecflowUI (requires Qt)	on
CMAKE_PREFIX_PATH	use to provide a path to dependent libraries that are installed in non-system locations. For example, if you have installed Qt in a non-system location, you should set the path in this variable.	
ENABLE_GUI	enable the build of ecflowview (requires X11 and motif)	on
ENABLE_ALL_TESTS	enable performance, migration, memory leak , and regression tests	off
ENABLE_SSL	Encrypted communication for user commands (experimental, from ecFlow release 4.5.0). Please see: Open ssl for more details.	off
ENABLE_SECURE_USER	password-based protection for user commands (experimental, from ecFlow release 4.5.0) Please see: Black list file (experimental) for more details.	off
BOOST_ROOT	where to find boost (if non-standard installation) If not specified cmake will look for an environment variable of the same name.	

The C++ compilers are chosen by CMake. (This can be overwritten by setting the environment variables *CXX* on the command line before you call *cmake*, to the preferred compiler).

Further, the variable *CMAKE_CXX_FLAGS* can be used to set compiler flags for optimisation or debugging.

cmake/ecbuild

```
cd $WK
mkdir build; cd build;

# Go with defaults, will build with CMAKE_BUILD_TYPE=Release and install to /usr/local
cmake ..

# build release with debug info
# cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo

# Override install prefix
# cmake .. -DCMAKE_INSTALL_PREFIX=/usr/local/apps/ecflow/4.14.0

# do NOT build the gui.
# cmake .. -DCMAKE_INSTALL_PREFIX=/usr/local/apps/ecflow -DCMAKE_BUILD_TYPE=Release -DENABLE_GUI=OFF

# ignore Wdeprecated-declarations compiler warning messages and do NOT build python api
# cmake .. -DCMAKE_CXX_FLAGS="-Wno-deprecated-declarations"  -DENABLE_PYTHON=OFF

# Use -j option to speed up compilation. Determine number of cpu's
CPUS=$(lscpu -p | grep -v '#' | wc -l)
make -j${CPUS}
make check
make install
```

If you experience problem with your installation, and need to fix your install of dependent libraries like QT,Python,Boost,gcc etc, then it is **VERY** important that you **delete** the build directory and start cmake build again. (This is because cmake keeps a cache of your configuration, and re-uses this unless the build directory is deleted).

Always remember to delete build directory if there is a change in system configuration

```
cd $WK
rm -rf build
mkdir build; cd build
cmake ..      # or use whatever cmake configuration you used before
```

To use the *ecFlow Python Api* , you need to add/change PYTHONPATH .

```
export PYTHONPATH=$PYTHONPATH:<prefix>/4.17.0/lib/python2.7/site-packages/ecflow
# If you used the default's then <prefix>=/usr/local
# otherwise you should use whatever you entered for -DCMAKE_INSTALL_PREFIX, hence in the examples above we
would have:
export PYTHONPATH=$PYTHONPATH:/usr/local/apps/ecflow/4.17.0/lib/python2.7/site-packages/ecflow
```

Installing ecFlow Python to a custom directory

The default install for ecFlow will install python(if it was enabled) under the directory given to CMAKE_INSTALL_PREFIX.

However, sometimes we may need to install the ecFlow python module to a different prefix. (starting with release 4.3.0)

This can be done using:

```
cd $WK/build  # change to the build directory
cmake -DCMAKE_INSTALL_PREFIX=/tmp/avi/custom/ecflow/4.17.0 -DCOMPONENT=python -P cmake_install.cmake -- make
install  # install python module under /tmp/avi/custom/ecflow/4.17.0
```

ecflow_ui: Make a list servers accessible to all users

The GUI used by ecFlow is called ecflow_ui. This is used to interact and visualize the ecFlow servers.

You can make the list of servers available for your users by:

- creating a file called **servers**
- The format of the server's file is very easy:

server file format

```
<server_name> <machine_name> <port>
```

An example might be:

servers file

```
server      machineX   3141
projectX    machineabc 4141
exp1        machineabc 4141
mars        bigmac     11031
```

- Copy this file to CMAKE_INSTALL_PREFIX/share/ecflow/. This makes the list of servers accessible to all users of ecflow_ui

```
cp servers /tmp/avi/custom/ecflow/4.17.0/share/ecflow/.
```