Installation guide for Escript and Finley

Release Revision: 2395 2.0Beta

Escript development team

April 20, 2009

Earth Systems Science Computational Centre (ESSCC)
The University of Queensland
Brisbane, Australia
Email: esys@esscc.uq.edu.au

CONTENTS

Introduction

This document describes how to install Escript/Finley on your computer. To learn how to use Escript/Finley please see the User guide or, for more detailed information, the API documentation.

Escript/Finley is developed primarily on Linux desktop systems, SGI ICE and MacOS. Binary distributions (discussed in Chapter 1) are available for the following platforms:

- Debian and Ubuntu Linux distributions (32-bit i686) (.deb package)
- Linux desktop systems with gcc (stand-alone bundle)
- MacOS X Leopard systems with gcc (stand-alone bundle)
- Windows systems with Visual Studio (stand-alone bundle)

Compilation from source is discussed in Chapter 2.

You can test your installation using the Python scripts in `examples.zip` or `examples.tar.gz` [1]. A simple test is

```
escript poission.py
```

It should produce a file called `u.xml` (which can be removed). If this is successful, then the main features of `esys.escript` have been sucessfully installed.

For visualisation we suggest `visit`[2] or `mayavi`[3].

For online help, see the site https://answers.launchpad.net/escript-finley.

[1] These will should either be in `escript.d/release/doc` or in the case of Debian, in `/usr/share/doc/escript`.
[2] https://wci.llnl.gov/codes/visit/
[3] http://mayavi.sourceforge.net

CHAPTER

ONE

Binary releases

Binary distributions (no compilation required) are available for the following operating systems.

- Linux - Section 1.1.
- MacOSX - Section 1.2
- Windows - Section 1.3

1.1 Linux binary installation

Escript/Finley can be installed as a stand-alone bundle, containing all the required tools. Alternatively, if we have a package for your distribution you can use the standard tools to install.

Please note, the current packages do not support OpenMP[1] or MPI[2]. If you need these features you may need to compile Escript/Finley from source (see Sections 2.2 and 2.3.1.)

For more information on using the `escript` command please see the User Guide.

1.1.1 Debian 5.0("Lenny")

Download the `escript.deb` file. (At time of writing we only produce debs for the i386 architecture.) Execute the following commands as root (you need to be in the directory containing the file).

```
dpkg --unpack escript.deb
aptitude install escript
```

If you use sudo this would be:

```
sudo dpkg --unpack escript.deb
sudo aptitude install escript
```

1.1.2 Ubuntu 8.10("Intrepid Ibex")

Since the installation process for Escript/Finley is pretty simple you should be able to use the same Lenny package for Ubuntu as well. Please notify the development team if this is not the case. Note that you will need to use the "sudo" instructions.

1.1.3 Stand-alone bundle

Download the bundle and decompress it.

[1] This is due to a bug related to gcc 4.3.2.
[2] Producing packages for MPI requires knowing something about your computer's configuration.

```
tar -xjf escript.tar.bz2
```

This will produce a directory called `stand`. You can rename or move it as is convienient to you. Test your installation by running:

```
stand/escript.d/bin/escript
```

You should get a normal python shell. If you wish to save on typing you can add `x/escript.d/bin` to your PATH variable (where x is the absolute path to your install).

1.2 MacOSX binary installation

Escript/Finley can be installed as a stand-alone bundle, containing all the required tools.

Please note, the current packages do not support OpenMP[3] or MPI[4]. If you need these features you may need to compile Escript/Finley from source (see Chapter 2.)

For more information on using the `escript` command please see the User Guide.

1.2.1 Stand-alone bundle Mac OS X 10.5.6 ("Leopard")

Download the bundle, double-click on it and copy `stand` anywhere you want. Then open a terminal[5] and type

```
cd x/stand
```

where *x* is the absolute path to your install. Now type

```
./install.sh
```

which will make some changes to your dynamic libraries to take into account your current location. Remember, you have to do this only once, when you just copied from our bundle. This procedure will not work if you decide to move your already working bundle into another folder. You have to copy it from our bundle again into your desired new location.

Test your installation by running:

```
 x/stand/escript.d/bin/escript
```

You should get a normal python shell. If you wish to save on typing you can add `x/escript.d/bin` to your PATH variable (where *x* is the absolute path to your install).

[3] This is due to a bug related to gcc 4.3.2.

[4] Producing packages for MPI requires knowing something about your computer's configuration.

[5] If you do not know how to open a terminal on Mac, then just type terminal in the spotlight (search tool on the top of the right corner) and once found just click on it.

1.3 Windows binary installation

There is no automated install/uninstall procedure for `esys.escript` on Windows at this time. However, the following procedure appears to work[6]

Please ensure you have the following software installed:

- pythonxy (http://www.pythonxy.com)
- numarray 1-5-2win32py2.5 for win32

From the escript zip file:

- copy the `esys` directory to your Python 2.5 site-packages folder (usually `C:\Python25\Lib\site-packages`).
- copy the `.dll` files from `esys_dlls` to a directory on your PATH. For example copy the directory to `C:\Python25\libs\esys_dlls` and add `C:\Python25\libs\esys_dlls` to your PATH.

[6] Thanks to Peter Hornby and Frederick Roger for this report.

CHAPTER

TWO

Source releases

Escript/Finley is known to compile and run on the following systems:

- Linux under gcc[1] - Section 2.3
- Linux under icc on SGI ICE 8200.
- MacOSX under gcc - Section 2.5

2.1 External dependencies

The following external packages are required in order to compile and run Escript/Finley. Where version numbers are specified, more recent versions can probably be subsituted. You can either try the standard/precompiled packages available for your operating system or you can download and build them from source. The advantage of using existing packages is that they will probably all work togther properly. You must take greater care if downloading sources separately.

- python-2.5.1 (http://python.org)
  - Python interpreter (You must compile with shared libraries.)
- numarray 1.5.2
  (http://www.stsci.edu/resources/software_hardware/numarray/numarray.html)
  - Arrays for python.
- boost-1.35 (http://www.boost.org)
  - Provides an interface between C++ and python.
- scons-0.989.5 (http://www.scons.org/)
  - a python-based alternative to "make".

The version numbers given here are not strict requirements, more recent (and in some cases older) versions will still work. The following packages should be sufficient (but not necessarily minimal) for Debian 5.0 ("Lenny"): python-dev, libboost1.35-dev, scons, python-numarray, g++.

These packages may be required for some of the optional capabilities of the system.

- netcdf-3.6.2 (http://www.unidata.ucar.edu/software/netcdf)
  - Used to save data sets in binary form for checkpoint/restart (must be compiled with -fPIC).
- vtk-5.0.4 (http://www.vtk.org)
  - This is used to save VTK files for visualization.
  - cmake-2.4.6 (http://www.cmake.org)
    - This is used to build VTK.

[1] There are some problems with OpenMP under gcc prior to version 4.3.2

  - mesa-7.0.3 (http://www.mesa3d.org)
    - Free OpenGL replacement used by VTK.

- mpich2-1.0.7 (http://www.mcs.anl.gov/research/projects/mpich2)
  - Parallelization with MPI.
- parmetis-3.1 (http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview)
  - Optimization of the stiffness matrix.
- MKL
  (http://www.intel.com/cd/software/products/asmo-na/eng/307757.htm)
  - Intel's Math Kernel Library for use with their c compiler.

The following packages might be useful for mesh generation:

- gmsh-2.2.0 (http://www.geuz.org/gmsh)
  - Mesh generation and viewing.
  - fltk-1.1.9 (http://www.fltk.org)
    - This is used to build gmsh
  - gsl-1.10 (http://www.gnu.org/software/gsl)
    - This is used to build gmsh
- triangle-1.6 (http://www.cs.cmu.edu/ quake/triangle.html)

Packages for visualization:

- mayavi-1.5 (http://mayavi.sourceforge.net)
  - MayaVi is referenced in our User Guide for viewing VTK files.
- visit-1.9 (https://wci.llnl.gov/codes/visit/)

2.2 Compilation

Throughout this section we will assume that the source code is uncompressed in a directory called trunk. You can call the directory anything you like, provided that you make the change before you compile.

You need to indicate where to find the external dependencies. Unless specified otherwise, all paths will be relative to the top level of the source. To do this, create a file in the `scons` directory called `x_options.py` where "x" is the name of your computer. As a starting point use one of the following:

- `scons/linux_options_example.py` (Linux desktop)
- `scons/mac_options_example.py` (MacOSX desktop)
- `ice_options_example.py` (SGI ICE 8200)
- `winxp_options_example.py` (Windows XP)

To actually compile (if you have n processors, then you can use `scons -j`n instead):

```
cd trunk
scons
```

As part of its output, scons will tell you the name of the options file it used as well as a list of features and whether they are enabled for your build.

If you require debug versions of the libraries, use:

```
scons usedebug=yes
```

A note about scons: if you recompile later with different options (eg leaving off usedebug), scons will revert to its default values. If you wish to make a change more permanent, then modify your options file.

You can install the binaries/libraries in a different location with:

```
scons prefix=some_dir
```

You can test your build using

```
scons all_tests
```

An alternative method is available for performing tests on OpenMP and MPI builds.

2.2.1 Compilation with OpenMP

You will need to consult your compiler documentation for the precise switches to use to enable OpenMP features. Once you know the options, modify the omp_optim, omp_debug and omp_libs variables in your options.py file.

For example, for gcc compilers which support OpenMP use.

```
omp_optim               = '-fopenmp'
omp_debug               = '-fopenmp'
omp_libs                = ['gomp']
```

Depending on your version, last change may not be required.

Then recompile.

```
scons useopenmp=yes
```

You can test your build on for instance 4 threads using

```
export ESCRIPT_NUM_THREADS=4
scons all_tests
```

2.2.2 Compilation with MPI

You will need to have MPI installed on your system. There are a number of implementations so we do not provide any specific advice here. You will need to modify the following variables in your options file.

- `mpi_flavour`
  which MPI implementation is used. Valid values are

  `MPT` SGI MPI implementation
  http://techpubs.sgi.com/library/manuals/3000/007-3687-010/pdf/007-3687-010.pdf

  `MPICH2` Argonne's MPICH vesrion 2 implementation
  http://www.mcs.anl.gov/research/projects/mpi/mpich2/

  `MPICH` Argonne's MPICH implementation
  http://www.mcs.anl.gov/research/projects/mpi/mpich1/

  `OPENMPI` Open MPI http://www.open-mpi.org/

  `INTELMPI` Intel's MPI http://software.intel.com/en-us/intel-mpi-library/

- `mpi_path`
  where to find `mpi.h`

- `mpi_lib_path`
  where to find libraries for mpi

- `mpi_libs`
  which libraries to link to.

Then compile with:

```
scons usempi=yes
```

As with debug and openmp, you can make this a more permanent setting by modifying your options file.

You can test your build on for instance 6 processors using

```
export ESCRIPT_NUM_NODES=6
scons usempi=yes all_tests
```

and on 6 processors with 4 threads each using

```
export ESCRIPT_NUM_THREADS=4
export ESCRIPT_NUM_NODES=6
scons usempi=yes all_tests
```

Alternatively, you can give a hostfile

```
export ESCRIPT_NUM_THREADS=4
export ESCRIPT_HOSTFILE=myhostfile
scons usempi=yes all_tests
```

Notice that depending on your MPI flavour it may be required to start a daemon before running the tests under MPI.

2.2.3 Difficulties

"Bad magic number"

Some reasons for this error message include:

- Using different versions of python when installing and running escript (Use `which python` and `python --version` to check)
- Using different versions of libraries (Make sure `LD_LIBRARY_PATH` has `/trunk/lib` listed first)
- Using different versions of python modules (Make sure `PYTHONPATH` has `/trunk/escript` directory listed first)

Another error we sometimes see is unsatisfied externals when trying to run a python script. This is usually due to not having `LD_LIBRARY_PATH` and `PYTHONPATH` set correctly so that you run with different libraries from the ones the code was compiled against. Check which libraries you are running against with `ldd lib/libfinley.so` and `ldd esys/finley/finleycpp.so`.

It is also possible that the person who compiled Escript/Finley used incompatible libraries. For example, if you run with Python2.4 but the software was compiled against Python2.5 then you will get unsatisfied externals or a large error message with a long traceback. Another case is when Boost or Numarray was compiled against the wrong Python library. To avoid these problems everyone (builder and user) must make certain they are using the same python libraries.

2.3 Installing from source for Linux

The following instructions assume you are running the `bash` shell. Comments are indicated with # characters.

Make sure you have the following installed:

- `g++` and associated tools.
- `make`
- `libXext.so`[2]
- `libxt.so`[3]

You will also need a copy of the Escript/Finley source code. If you retrieved the source using subversion, don't forget that one can use the export command instead of checkout to get a smaller copy. For additional visualisation functionality see Section 2.4.

These instructions will produce the following structure:

- `stand`:
  - `escript.d`
  - `packages`
  - `package_src`
  - `build`
  - `doc`

The build directory can be removed when you are finished.

```
mkdir stand
cd stand
export PKG_ROOT=`pwd`/packages
```

Copy compressed source bundles into `stand/package_src`. Copy documentation files into `doc`.

```
mkdir packages
mkdir build
cd build
tar -jxf ../package_src/Python-2.6.1.tar.bz2
tar -jxf ../package_src/boost_1_37_0.tar.bz2
tar -jxf ../package_src/MesaLib-7.2.tar.bz2
tar -zxf ../package_src/netcdf-4.0.tar.gz
tar -zxf ../package_src/vtk-5.2.1.tar.gz
tar -zxf ../package_src/vtkdata-5.2.1.tar.gz
tar -zxf ../package_src/numarray-1.5.2.tar.gz
tar -zxf ../package_src/cmake-2.6.3.tar.gz
tar -zxf ../package_src/scons-1.2.0.tar.gz
```

Build python.

```
cd Python*
./configure --prefix=$PKG_ROOT/python-2.6.1 --enable-shared 2>&1 \
  | tee tt.configure.out
make install 2>&1 | tee tt.make.out

cd ..

export PATH=$PKG_ROOT/python/bin:$PATH
export PYTHONHOME=$PKG_ROOT/python
```

[2] In Debian this is in the libXext-dev package.
[3] In Debian this is in the libxt-dev package.

```
export LD_LIBRARY_PATH=$PKG_ROOT/python/lib:$LD_LIBRARY_PATH

pushd ../packages
ln -s python-2.6.1/ python
popd
```

Run python to make sure it works. Now build numarray.

```
cd numarray-1.5.2

python setup.py install \
 --gencode --install-lib=$PKG_ROOT/numarray-1.5.2/lib \
 --install-headers=$PKG_ROOT=$PKG_ROOT/numarray-1.5.2/include/numarray \
   2>&1 | tee tt.install.out


export PYTHONPATH=$PKG_ROOT/numarray/lib:$PYTHONPATH
cd ..
pushd ../packages
ln -s numarray-1.5.2 numarray
popd
```

Now we build scons.

```
cd scons-1.2.0
python setup.py install --prefix=$PKG_ROOT/scons-1.2.0

export PATH=$PKG_ROOT/scons/bin:$PATH
cd ..
pushd ../packages
ln -s scons-1.2.0 scons
popd
```

...Boost libraries ...

```
cd boost_1_37_0

./configure --prefix=$PKG_ROOT/boost_1_37_0 --with-python-root=$PKG_ROOT/python \
  --with-python-version=2.6 --with-libraries=python

make
make install
ln -s $PKG_ROOT/boost_1_37_0 $PKG_ROOT/boost
export LD_LIBRARY_PATH=$PKG_ROOT/boost/lib:$LD_LIBRARY_PATH
cd ..
pushd ../packages
ln -s boost_1_37_0 boost
popd
```

... and netcdf.

```
cd netcdf-4.0
CFLAGS="-O2 fPIC -Df2cFortran" CXXFLAGS="-O2 fPIC -Df2cFortran" \
FFLAGS="-O2 fPIC -Df2cFortran" FCFLAGS="-O2 fPIC -Df2cFortran" \
./configure --prefix=$PKG_ROOT/netcdf-4.0

make -j2
make install

export LD_LIBRARY_PATH=$PKG_ROOT/netcdf/lib:$LD_LIBRARY_PATH
cd ..
pushd ../packages
ln -s netcdf-4.0 netcdf
popd
```

CMake and Mesa are required for VTK.

```
cd cmake-2.6.3
./configure --prefix=$PKG_ROOT/cmake-2.6.3 2>&1 | tee tt.configure
make -j 4
make install

export PATH=$PKG_ROOT/cmake/bin:$PATH
cd ..
pushd ../packages
ln -s cmake-2.6.3 cmake
popd
```

These instructions do not compile MesaDemos or GLUT. If you need to check if Mesa compiled correctly, then the demos are a good test.

```
cd Mesa-7.2
./configure --prefix=$PKG_ROOT/mesa-7.2 --enable-gl-osmesa --with-driver=xlib

make -j 4
make install

export LD_LIBRARY_PATH=$PKG_ROOT/mesa:$LD_LIBRARY_PATH
cd ..
pushd ../packages
ln -s mesa-7.2 mesa
popd
```

```
cd VTK
cmake .

#Edit the CMakeCache and make the following changes:
#(Please replace .... with an absolute path to the stand directory)

#-----------------

BUILD_EXAMPLES should be OFF
BUILD_SHARED_LIBS should be ON

CMAKE_INSTALL_PREFIX    ..../stand/packages/vtk-5.2.1
CMAKE_VERBOSE_MAKEFILE  TRUE

#check PYTHON_EXECUTABLE is correct.
#but it seems to be when I went through these steps

VTK_OPENGL_HAS_OSMESA   TRUE
VTK_USE_64BIT_IDS       ON
# That last one is marked as "May cause some bugs" in the original instructions

VTK_WRAP_PYTHON ON
VTK_USE_MANGLED_MESA    OFF

#--------------------

cmake .
#It won't work but it will put some variables in that you need.

#Edit CMakeCache again and make the following changes

#----------------

VTK_USE_TK      OFF

OSMESA_INCLUDE_DIR      ..../stand/packages/mesa/include
```

```
OSMESA_LIBRARY  ..../stand/packages/mesa/lib/libOSMesa.so

PYTHON_INCLUDE_PATH     ..../stand/packages/python/include/python2.6

PYTHON_LIBRARY  ..../stand/packages/python/lib/libpython2.6.so

OPENGL_INCLUDE_DIR      ..../stand/packages/mesa/include

OPENGL_gl_LIBRARY       ..../stand/packages/mesa/lib/libGL.so

#---------------

cmake .
make
make install


cd ../../packages
ln -s vtk-5.2.1 vtk
cd ..
```

Now copy the Escript/Finley source into an `escript.d` directory in `stand`.

2.3.1 Compiling escript

Change to the directory containing your escript source (`escript.d`), then:

```
cd scons
cp linux_options_example.py YourMachineName_options.py

#edit the options file and make the following changes:
#-----------------------------------------------------------------
declare a PKG_ROOT variable at the top of the file eg:
PKG_ROOT='/home/jfenwick/stand/packages'

python_path             = PKG_ROOT+'python/include/python2.6'
python_lib_path         = PKG_ROOT+'python/lib'
python_libs             = 'python2.6'

boost_path              = PKG_ROOT+'boost/include/boost-1_37'
boost_lib_path          = PKG_ROOT+'boost/lib'
boost_libs              = ['boost_python-gcc43-mt']
# You could simlink the boost python library to give a shorter
# name but it's not worth it

usevtk          = 'yes'
#-----------------------------------------------------------------

ln -s $PKG_ROOT/vtk-5.2.1 $PKG_ROOT/vtk

Modify /scripts/finley_wrapper_template

STANDALONE=1

#Check to make sure the paths in the if [ $STANDALONE == 1 ]
# Section are correct

#-----------------------------------------------------------------

scons bin/escript

#start a new terminal
```

```
cd stand
export PATH=`pwd`/packages/scons/bin:$PATH
cd escript.d
eval `bin/escript -e`
scons
```

If you wish to test your build, then you can do the following. Note this may take a while if you have a slow processor and/or less than 1Gb of RAM.

```
scons all_tests
```

2.3.2 Cleaning up

Once you are satisfied, the `escript.d/build` and `$PKG_ROOT/build` directories can be removed.

If you *really* want to save space and do not wish to be able to edit or recompile escript, you can remove the following:

- From the `escript.d` directory:
  - Everything except: `bin`, `include`, `lib`, `esys`, `README_LICENSE`.
  - Hidden files, which can be removed using

    ```
    find . -name .?* | xargs rm -rf
    ```

    in the `escript.d` directory.
- from the `packages` directory:
  - `scons`, `scons-1.2.0`, `cmake-2.6.3` and `cmake`
- `package_src`[4].

Please note that removing all these files may make it more difficult for us to diagnose problems.

2.4 Additional Functionality

To perform visualisations you will need some additional tools. Since these do not need to be linked with any of the packages above, you can install versions available for your system, or build them from source.

- `ppmtompeg` and `jpegtopnm` from the `netpbm` suite. - To build from source you would also need `libjpeg` and its headers as well as `libpng`[5] and its headers.
- A tool to visualise VTK files. For example Mayavi or Visit.

[4] Do not remove this if you intend to redistribute.
[5] libpng requires zlib to build

2.5 Installing from source for MacOSX

First of all before you start installing from source you will need Mac OS X development tools installed on your Mac. This will ensure that you have the following installed:

- `g++` and associated tools.
- `make`

Here are the instruction on how to install them.

1. Insert the Mac OS X v10.5 (Leopard) DVD.
2. Double-click on XcodeTools.mpkg, located inside Optional Installs/Xcode Tools.
3. Follow the instructions in the Installer.
4. Authenticate as the administrative user. The first user you create when setting up Mac OS X has administrator privileges by default.

You will also need a copy of the Escript/Finley source code. If you retrieved the source using subversion, don't forget that one can use the export command instead of checkout to get a smaller copy. For additional visualisation functionality see Section 2.6.

These instructions will produce the following structure:

- `stand`:
  - `escript.d`
  - `packages`
  - `package_src`
  - `build`
  - `doc`

The build directory can be removed when you are finished.

The following instructions assume you are running the `bash` shell. Comments are indicated with # characters.

Open a terminal [6] and type

```
mkdir stand
cd stand
export PKG_ROOT=`pwd`/packages
```

Copy compressed source bundles into `stand/package_src`. Copy documentation files into `doc`.

```
mkdir packages
mkdir build
cd build
tar -jxf ../package_src/Python-2.6.1.tar.bz2
tar -jxf ../package_src/boost_1_38_0.tar.bz2
tar -jxf ../package_src/MesaLib-7.2.tar.bz2
tar -zxf ../package_src/netcdf-4.0.tar.gz
tar -zxf ../package_src/vtk-5.2.1.tar.gz
tar -zxf ../package_src/vtkdata-5.2.1.tar.gz
tar -zxf ../package_src/numarray-1.5.2.tar.gz
tar -zxf ../package_src/cmake-2.6.3.tar.gz
tar -zxf ../package_src/scons-1.2.0.tar.gz
```

Build python.

[6] If you do not know how to open a terminal on Mac, then just type terminal in the spotlight (search tool on the top of the right corner) and once found just click on it.

```
cd Python*

./configure --prefix=$PKG_ROOT/python-2.6.1 \
  --exec-prefix=$PKG_ROOT/python-2.6.1 --enable-shared \
  --enable-framework=$PKG_ROOT/python-2.6.1 2>&1 | tee tt.configure.out

make -j2

make install 2>&1 | tee tt.make.out

cp ../../packages/python-2.6.1/Python.framework/Versions/2.6/Python \
     ../../packages/python-2.6.1/lib/libpython2.6.dylib

cp ../../packages/python-2.6.1/Python.framework/Versions/2.6/Python \
      ../../packages/python-2.6.1/lib/libpython.dylib

cp ../../packages/python-2.6.1/include/python2.6/pyconfig.h \
     ../../packages/python-2.6.1/Python.framework/Headers/

cd ..

export PATH=$PKG_ROOT/python/bin:$PATH
export PYTHONHOME=$PKG_ROOT/python
export LD_LIBRARY_PATH=$PKG_ROOT/python/lib:$LD_LIBRARY_PATH

pushd ../packages
ln -s python-2.6.1/ python
popd
```

Run python to make sure (check the version number) it works. Now build numarray.

```
cd numarray-1.5.2

python setup.py install \
 --gencode --install-lib=$PKG_ROOT/numarray-1.5.2/lib \
 --install-headers=$PKG_ROOT=$PKG_ROOT/numarray-1.5.2/include/numarray \
   2>&1 | tee tt.install.out


export PYTHONPATH=$PKG_ROOT/numarray/lib:$PYTHONPATH
cd ..
pushd ../packages
ln -s numarray-1.5.2 numarray
popd
```

Now we build scons.

```
cd scons-1.2.0
python setup.py install --prefix=$PKG_ROOT/scons-1.2.0

export PATH=$PKG_ROOT/scons/bin:$PATH
cd ..
pushd ../packages
ln -s scons-1.2.0 scons
popd
```

...Boost libraries ...

```
cd boost_1_38_0

./configure --prefix=$PKG_ROOT/boost_1_38_0 --with-python-root=$PKG_ROOT/python \
  --with-python-version=2.6 --with-libraries=python

make -j2
make install
```

```
ln -s $PKG_ROOT/boost_1_38_0 $PKG_ROOT/boost
export LD_LIBRARY_PATH=$PKG_ROOT/boost/lib:$LD_LIBRARY_PATH
cd ..
pushd ../packages
ln -s boost_1_38_0 boost
popd
```

... and netcdf.

```
cd netcdf-4.0
CFLAGS="-O2 fPIC -Df2cFortran" CXXFLAGS="-O2 fPIC -Df2cFortran" \
FFLAGS="-O2 fPIC -Df2cFortran" FCFLAGS="-O2 fPIC -Df2cFortran" \
./configure --prefix=$PKG_ROOT/netcdf-4.0

make -j2
make install

export LD_LIBRARY_PATH=$PKG_ROOT/netcdf/lib:$LD_LIBRARY_PATH
cd ..
pushd ../packages
ln -s netcdf-4.0 netcdf
popd
```

CMake and Mesa are required for VTK.

```
cd cmake-2.6.3
./configure --prefix=$PKG_ROOT/cmake-2.6.3 2>&1 | tee tt.configure
make -j 4
make install

export PATH=$PKG_ROOT/cmake/bin:$PATH
cd ..
pushd ../packages
ln -s cmake-2.6.3 cmake
popd
```

These instructions do not compile MesaDemos or GLUT. If you need to check if Mesa compiled correctly, then the demos are a good test.

```
cd Mesa-7.2
./configure --prefix=$PKG_ROOT/mesa-7.2 --enable-gl-osmesa

make -j 4
make install

export LD_LIBRARY_PATH=$PKG_ROOT/mesa:$LD_LIBRARY_PATH
cd ..
pushd ../packages
ln -s mesa-7.2 mesa
popd
```

```
cd VTK
cmake .

#Edit the CMakeCache and make the following changes:
#(Please replace .... with an absolute path to the stand directory)

#-----------------

BUILD_EXAMPLES should be OFF
BUILD_SHARED_LIBS should be ON

CMAKE_INSTALL_PREFIX    ..../stand/packages/vtk-5.2.1
CMAKE_VERBOSE_MAKEFILE  TRUE
```

```
#check PYTHON_EXECUTABLE is correct.
#but it seems to be when I went through these steps

VTK_OPENGL_HAS_OSMESA   TRUE
VTK_USE_64BIT_IDS       ON
# That last one is marked as "May cause some bugs" in the original instructions

VTK_WRAP_PYTHON ON
VTK_USE_MANGLED_MESA    OFF

#--------------------

cmake .
#It won't work but it will put some variables in that you need.

#Edit CMakeCache again and make the following changes

#----------------

VTK_USE_TK      OFF

OSMESA_INCLUDE_DIR      ..../stand/packages/mesa/include

OSMESA_LIBRARY  ..../stand/packages/mesa/lib/libOSMesa.dylib

PYTHON_INCLUDE_PATH     ..../stand/packages/python/include/python2.6

PYTHON_LIBRARY  ..../stand/packages/python/lib/libpython2.6.dylib

OPENGL_INCLUDE_DIR      ..../stand/packages/mesa/include

OPENGL_gl_LIBRARY       ..../stand/packages/mesa/lib/libGL.dylib

#----------------

cmake .
make
make install


cd ../../packages
ln -s vtk-5.2.1 vtk
cd ..
```

Now copy the Escript/Finley source into an `escript.d` directory in `stand`.

```
cd scons
cp mac_options_example.py YourMachineName_options.py

#edit the options file and make the following changes:
#-----------------------------------------------------------------
declare a PKG_ROOT variable at the top of the file eg:
PKG_ROOT='/Users/artak/stand/packages'

python_path             = PKG_ROOT+'python/include/python2.6'
python_lib_path         = PKG_ROOT+'python/lib'
python_libs             = 'python2.6'

boost_path              = PKG_ROOT+'boost/include/boost-1_38'
boost_lib_path          = PKG_ROOT+'boost/lib'
boost_libs              = ['boost_python-gcc43-mt']
# You could simlink the boost python library to give a shorter
# name but it's not worth it
```

```
usevtk          = 'yes'
#-----------------------------------------------------------------

ln -s $PKG_ROOT/vtk-5.2.1 $PKG_ROOT/vtk

Modify /scripts/finley_wrapper_template

STANDALONE=1

#Check to make sure the paths in the if [ $STANDALONE == 1 ]
# Section are correct

#-----------------------------------------------------------------

scons bin/escript

#start a new terminal
cd stand
export PATH=`pwd`/packages/scons/bin:$PATH
cd escript.d
eval `bin/escript -e`
scons
```

If you wish to test your build, then you can do the following. Note this may take a while if you have a slow processor and/or less than 1Gb of RAM.

```
scons all_tests
```

Once you are satisfied, the `build` and `$PKG_ROOT/build` directories can be removed. Within the `packages` directory, the `scons`, `scons-1.2.0`, `cmake-2.6.3` and `cmake` entries can also be removed. If you are not redistributing this bundle you can remove `$PKG_ROOT/package_src`.

If you do not plan to edit or recompile the source you can remove it. The only entries which are required in `escript.d` are:

- `bin`
- `esys`
- `include`
- `lib`
- `README_LICENSE`

Hidden files can be removed with

```
find . -name .?* | xargs rm -rf
```

2.6 Additional Functionality

To perform visualisations you will need some additional tools. Since these do not need to be linked with any of the packages above, you can install versions available for your system, or build them from source.

- `ppmtompeg` and `jpegtopnm` from the `netpbm` suite. - To build from source you would also need `libjpeg` and its headers as well as `libpng`[7] and its headers.
- A tool to visualise VTK files. For example Mayavi or Visit.

[7] libpng requires zlib to build

2.7 Installing from source for MacOSX using Macports

As we mentioned in the Section 2.5, before you start installing from source you will need Mac OS X development tools installed on your Mac.

If you do not have Macports already, please install Macports from www.macports.org. You can also install porticus (GUI for Macports).

Once you have Macports working Install boost using porticus or from terminal

```
sudo port install boost@1.35.0_2+complete
```

Sometimes this fails due to unknown reasons, but to overcome this problem you need to run

```
sudo port clean boost@1.35.0_2+complete

sudo port install boost@1.35.0_2+complete
```

Download scons source scons-0.98.5.tar.gz from www.scons.org.

```
tar xfz scons-0.98.5.tar.gz
cd scons-0.98.5
python setup.py install
```

Note: Do not try to install scons using porticus or sudo port install scons, because if automatically installs another python version and you are likely run into problems with different python versions.

Download numarray-1.5.2.tar.gz from
http://www.stsci.edu/resources/software_hardware/numarray/numarray.html.

```
tar xfz numarray-1.5.2.tar.gz
cd numarray-1.5.2
python setup.py install --gencode  2>&1 | tee tt.install.out
```

You can ran a test to check numarray installation by

```
python
import numarray.testall as testall
testall.test()
```

Install netcdf, gsl and fltk using Macports

```
sudo port install netcdf
sudo port install gsl
sudo port intall fltk
```

Note: If this fails, download and install from sources.

Downlaod gmsh-2.2.3-source.tar and install from sources.

```
./configure --with-gsl-prefix=/opt/local/  --with-fltk    --prefix=/usr/local/
```

Note: if you install using porticus or sudo port it automaticall installs in `/opt/local/`, but if you install from sources it installs in /usr/local. So, make sure these paths are right.

```
sudo make -j2
sudo make install
```

Download and install Mesa-7.0.3 (required for VTK) from sources

```
tar xjf MesaDemos-7.0.3.tar.bz2
tar xjf MesaGLUT-7.0.3.tar.bz2
tar xjf MesaLib-7.0.3.tar.bz2
cd Mesa-7.0.3

sudo make -j 2
make install
```

Install vtk-5.0.4 from source. If you install from ports it won't configure to use shared libraries. Once you untar it you will have (assume user is john) /Users/john/Downloads/VTK folder, then run following:

```
sudo mkdir /usr/local/VTKBuild/
cd /usr/local/VTKBuild/
sudo ccmake /Users/john/Downloads/VTK/
# It will create CMakeCache.txt. Make sure  you use the following configurations.

#       Advanced options                          ON
#       BUILD_EXAMPLES                            ON
#       BUILD_SHARED_LIBS                         ON
#       VTK_WRAP_PYTHON                           ON
#       CMAKE_VERBOSE_MAKEFILE                    TRUE
#       VTK_OPENGL_HAS_OSMESA                     ON
#       VTK_USE_MANGLED_MESA                      OFF
#       VTK_USE_OFFSCREEN                         OFF

sudo make -j2
sudo make install
```

Note: you need to set following ENV variables into your `/Users/john/.profile` for VTK to work:

```
export LD_LIBRARY_PATH = /usr/local/VTKBuild/bin: \
             /usr/local/VTKBuild/bin:${LD_LIBRARY_PATH}
export PYTHONPATH = /usr/local/VTKBuild/Wrapping/Python:\
            /usr/local/VTKBuild/bin:${PYTHONPATH}

#     For testing you can run:
python
import vtk
```

Left only to install triangle and netpbm (required for ppmtompeg) using Macports

```
sudo port install triangle
sudo port install netpbm
```