Installation guide for Escript and Finley

Release 3.0
(r2601)

Escript development team

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

CONTENTS

CHAPTER

ONE

Introduction

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

Escript/Finley is primarily developed on Linux desktop, SGI ICE and MacOS X systems. It is distributed in two forms:

1. Binary bundles – these are great for first time users or for those who want to start using Escript/Finley immediately
2. Source bundles – these require compilation and should be used if the binary bundles don't work on the target machine or if extra functionality is required such as MPI parallelisation.

The binary bundles are currently 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)

Hopefully, a Windows version(stand-alone) of this release will be available soon.

See Chapter 2 for instructions on how to set these up and run Escript/Finley. If you choose to compile from source your options are to

- install dependencies (e.g. using your package manager) and only compile Escript/Finley, OR
- compile everything from source.

Compiling Escript/Finley when its dependencies are already installed is discussed in Chapter 3. To compile Escript/Finley and all dependencies from source please see Chapter 4. The latter option takes a significant amount of time and is only required if the versions of the dependent libraries available on your system do not work with Escript/Finley.

Once everything is installed you can test your installation using the Python scripts in `examples.zip` or `examples.tar.gz`[1]. Unpack the examples and try to run the following from a terminal:

```
escript poission.py
```

If this produces a VTK file called `u.vtu` then you are likely to have a functional Escript/Finley installation. You can try and visualize the VTK data or delete the file. For visualization we suggest using `VisIt`[2] or `MayaVi`[3] which are both freely available.

See the site https://answers.launchpad.net/escript-finley for online help.

[1] These 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

TWO

Binary releases

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

- Linux – Section 2.1
- MacOS X – Section 2.2

2.1 Linux binary installation

Escript/Finley can be installed as a stand-alone bundle, containing all the required dependencies. Alternatively, if we have a package for your distribution you can use the standard tools to install. Please note, however, that the current binary packages do not support OpenMP[1] or MPI[2]. If you need these features you may need to compile Escript/Finley from source (see Section 3.2 and Section 4.1.3.)

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

If you are using Debian (5.0 - "Lenny") or Ubuntu (8.10-"Intrepid Ibex", 9.04-"Jaunty Jackalope") then see Section 2.1.1. For other Linux distributions refer to Section 2.1.2.

2.1.1 Debian 5.0("Lenny"), Ubuntu 8.10("Intrepid Ibex") or 9.04("Jaunty Jakalope")

At the time of this writing we only produce deb's for the i386 and amd64 architectures. The package file will be named `escript-X-D_A.deb` where `X` is the version, `D` is either "`lenny`" or "`jaunty`" and `A` is the architecture. For example, `escript-3.0-1-lenny_amd64.deb` would be the file for lenny (and intrepid) for 64bit processors. To install Escript/Finley download the appropriate `.deb` file and 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 (for example on Ubuntu) enter the following instead:

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

This should install Escript/Finley and its dependencies on your system. Please notify the development team if something goes wrong.

2.1.2 Stand-alone bundle

If there is no package available for your distribution, you may be able to use one of our stand alone bundles. These come in two parts: escript itself (`escript_3.0_i386.tar.bz2`) and a group of required programs

[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.

(escript-support_3.0_i386.tar.bz2). For 64-bit Intel and Amd processors substitute amd64 for i386.

```
tar -xjf escript-support_3.0_i386.tar.bz2
tar -xjf escript_3.0_i386.tar.bz2
```

This will produce a directory called stand which contains a stand-alone version of Escript/Finley and its dependencies. You can rename or move it as is convenient to you, no installation is required. Test your installation by running:

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

This should give you a normal python shell. If you wish to save on typing you can add x/stand/escript.d/bin[3] to your PATH variable (where x is the absolute path to your install).

[3] or whatever you renamed stand to.

2.2 MacOS X 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[4] or MPI[5]. If you need these features you may need to compile Escript/Finley from source (see Chapter 4.)

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

2.2.1 Stand-alone bundle MacOS X 10.5.6 ("Leopard")

You will need to download both escript (`escript_3.0_osx.dmg`) and the support files (`escript-support_3.0_osx.dmg`).

- Create a folder to hold escript (no spaces in the name please).
- Open the `.dmg` files and copy the contents to the folder you just created.

To use escript, open a terminal[6] and type

```
cd x/escript.d/bin/escript
```

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

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).

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

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

[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.

CHAPTER

THREE

Building escript from source

This chapter describes how to build Escript/Finley from source assuming that the dependencies are already installed (for example using precompiled packages for your OS). Section 3.1 describes the dependencies, while Section 3.2 gives the compile instructions.

If you would prefer to build all the dependecies from source in the escript-support packages please see Chapter 4. Escript/Finley is known to compile and run on the following systems:

- Linux using gcc[1]
- Linux using icc on SGI ICE 8200
- MacOS X using gcc
- Windows XP using the Visual C compiler (we do not specifically discuss Windows builds in this guide).

3.1 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 are more likely to work together 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.)
- numpy 1.1.0 (http://numpy.scipy.org)
  - Arrays for python
- boost-1.35 (http://www.boost.org)
  - Interface between C++ and Python
- scons-0.989.5 (http://www.scons.org/)
  - Python-based alternative to "make".

The version numbers given here are not strict requirements, more recent (and in some cases older) versions are very likely to work. The following packages should be sufficient (but not necessarily minimal) for Debian 5.0 ("Lenny"): python-dev, libboost-python1.35-dev, scons, python-numpy, 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)

[1] There are some problems with OpenMP under gcc prior to version 4.3.2. Also do not link the gomp library with gcc 4.3.3 - it causes problems.

- vtk-5.0.4 (http://www.vtk.org)
  - Used to save VTK files for visualization

  – cmake-2.4.6 (http://www.cmake.org)
    - Required to build VTK

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

- netpbm (http://netpbm.sourceforge.com)
  - Tools for producing movies from images

- 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)
    - Required to build gmsh

  – gsl-1.10 (http://www.gnu.org/software/gsl)
    - Required to build gmsh

- triangle-1.6 (http://www.cs.cmu.edu/ quake/triangle.html)
  - Two-dimensional mesh generator and Delaunay triangulator.

Packages for visualization:

- mayavi-1.5 (http://mayavi.sourceforge.net)
  - MayaVi is referenced in our User's Guide for viewing VTK files

- visit-1.11.2 (https://wci.llnl.gov/codes/visit/)
  - A featureful visualization system with movie-making capabilities.

3.2 Compilation

Throughout this section we will assume that the source code is uncompressed in a directory called `escript.d`. 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. To do this, create a file in the `escript.d/scons` directory called `x_options.py` where "x" is the name of your computer (output of the `hostname` command). Please note that if your hostname has non-alphanumeric characters in it (eg -) you need to replace them with underscores. For example the options file for `bob-desktop` would be named `bob_desktop_options.py`.

From now on all paths will be relative to the top level of the source. As a starting point copy the contents one of the following files :

- `scons/linux_options_example.py` (Linux desktop)
- `scons/mac_options_example.py` (MacOS X desktop)
- `scons/ice_options_example.py` (SGI ICE 8200)

- scons/winxp_options_example.py (Windows XP)

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

```
cd escript.d
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 (e.g. leaving out 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
```

Grab a coffee or two while the tests compile and run. An alternative method is available for performing tests on OpenMP and MPI builds.

3.2.1 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, the last change may not be required. If you're unsure try without the gomp library first and add it if you get linker errors.

Then recompile.

```
 scons useopenmp=yes
```

You can test your build, e.g. using 4 threads by issuing

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

3.2.2 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 version 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.

To test your build using 6 processors enter:

```
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
```

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

3.2.3 Difficulties

"Bad magic number"

This error usually indicates that the version of python used to run escript differs from the version used when installing escript (Use `which python` and `python --version` to check).

It is also possible that incompatible libraries were used when compiling Escript/Finley. 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 both builder and user must ensure they are using the same python libraries.

OpenMP builds segfault running examples

One known cause for this is linking the `gomp` library with escript built using gcc 4.3.3. While you need the `-fopenmp` switch you should not need to link `gomp`.

CHAPTER

FOUR

Building escript and dependencies from source

This chapter describes how to build escript and its dependencies from the source code in the escript support packages. You can also use these instructions if you have gathered the various sources yourself.

4.1 Installing from source for Linux

4.1.1 Preliminaries

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`

To compile matplotlib you will also need the following[1] (if your distribution separates development files, make sure to get the development packages):

- `freetype2`
- `zlib`
- `libpng`

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 visualization functionality see Section 4.3.

These instructions will produce the following directory structure:

```
stand
    escript.d
    pkg
    pkg_src
    build
    doc
```

[1] For Debian and Ubuntu users, installing `libfreetype6-dev` and `libpng-dev` will be sufficient.

Before you start copy the Escript/Finley source into the `escript.d` directory. The following instructions refer to software versions in the `escript-support-3-src` bundle. If you download your own versions of those packages substitute their version numbers and names as appropriate. There are a number of uses of the `make` command in the following instructions. If your computer has multiple cores/processors you can speed up the compilation process by adding -j 2 after the make command. For example to use all processors on a computer with 4 cores:

```
make
```

becomes

```
make -j 4
```

```
mkdir stand
cd stand
mkdir build doc pkg pkg_src
export PKG_ROOT=$(pwd)/pkg
```

4.1.2 Building the dependencies

Copy the compressed sources for the packages into `stand/pkg_src`. If you are using the support bundles, decompress them in the stand directory:

```
tar -xjf escript-support-3-src.tar.bz2
```

Copy documentation files into `doc` then unpack the archives:

```
cd build
tar -jxf ../pkg_src/Python-2.6.2.tar.bz2
tar -jxf ../pkg_src/boost_1_39_0.tar.bz2
tar -zxf ../pkg_src/scons-1.2.0.tar.gz
tar -zxf ../pkg_src/numpy-1.3.0.tar.gz
tar -zxf ../pkg_src/netcdf-4.0.tar.gz
tar -zxf ../pkg_src/matplotlib-0.98.5.3.tar.gz
```

- Build Python:

  ```
  cd Python*
  ./configure --prefix=$PKG_ROOT/python-2.6.2 --enable-shared 2>&1 \
    | tee tt.configure.out
  make
  make install 2>&1 | tee tt.make.out

  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 ../pkg
  ln -s python-2.6.2/ python
  popd
  ```

  Run the new python executable to make sure it works.

- Now build NumPy:

  ```
  cd numpy-1.3.0
  python setup.py build
  python setup.py install --prefix $PKG_ROOT/numpy-1.3.0
  cd ..
  pushd ../pkg
  ln -s numpy-1.3.0 numpy
  popd
  export PYTHONPATH=$PKG_ROOT/numpy/lib/python2.6/site-packages:$PYTHONPATH
  ```

- Next 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 ../pkg
ln -s scons-1.2.0 scons
popd
```

- The Boost libraries...:

```
pushd ../pkg
mkdir boost_1_39_0
ln -s boost_1_39_0 boost
popd
cd boost_1_39_0
./bootstrap.sh --with-libraries=python --prefix=$PKG_ROOT/boost
./bjam
./bjam install --prefix=$PKG_ROOT/boost --libdir=$PKG_ROOT/boost/lib
export LD_LIBRARY_PATH=$PKG_ROOT/boost/lib:$LD_LIBRARY_PATH
cd ..
pushd ../pkg/boost/lib/
ln *.so.* libboost_python.so
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
make install

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

- Finally matplotlib:

```
cd matplotlib-0.98.5.3
python setup.py build
python setup.py install --prefix=$PKG_ROOT/matplotlib-0.98.5.3
cd ..
pushd ../pkg
ln -s matplotlib-0.98.5.3 matplotlib
popd
cd ..
```

4.1.3 Compiling escript

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

```
cd escript.d/scons
cp linux_standalone_options_example.py YourMachineName_options.py

echo $PKG_ROOT
```

Where YourMachineName is the name of your computer as returned by the hostname command. If the name contains non-alphanumeric characters, then you will need to replace them with underscores. For example the options file for bob-desktop would be named bob_desktop_options.py.

Edit the options file and put the value of PKG_ROOT between the quotes in the PKG_ROOT= line. For example:

```
PKG_ROOT="/home/bob/stand/pkg"
```

```
cd ../bin
```

Modify the STANDALONE line of escript to read:

STANDALONE=1

Start a new terminal and go to the stand directory.

```
export PATH=$(pwd)/pkg/scons/bin:$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
```

4.1.4 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/Finley, 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 pkg directory:
  - scons, scons-1.2.0, cmake-2.6.3 and cmake
- package_src[2].

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

4.2 Installing from source for MacOS X

Before you start installing from source you will need MacOS X development tools installed on your Mac. This will ensure that you have the following available:

- g++ and associated tools.
- make

Here are the instructions on how to install these.

[2] Do not remove this if you intend to redistribute.

1. Insert the MacOS X 10.5 (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 MacOS 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 visualization functionality see Section 4.3.

These instructions will produce the following directory structure:

- stand:
  - escript.d
  - pkg
  - pkg_src
  - build
  - doc

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

Open a terminal [3] and type

```
mkdir stand
cd stand
export PKG_ROOT=‘pwd‘/pkg
```

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

```
mkdir packages
mkdir build
cd build
tar -jxf ../pkg_src/Python-2.6.2.tar.bz2
tar -jxf ../pkg_src/boost_1_39_0.tar.bz2
tar -zxf ../pkg_src/scons-1.2.0.tar.gz
tar -zxf ../pkg_src/numpy-1.3.0.tar.gz
tar -zxf ../pkg_src/netcdf-4.0.tar.gz
tar -zxf ../pkg_src/matplotlib-0.98.5.3.tar.gz
```

- Build python:

  ```
  cd Python*
  ./configure --prefix=$PKG_ROOT/python-2.6.2 --enable-shared 2>&1 \
    | tee tt.configure.out
  make
  make install 2>&1 | tee tt.make.out

  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 ../pkg
  ln -s python-2.6.2/ python
  popd
  ```

[3] 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.

Run the new python executable to make sure it works.

- Now build NumPy:

```
cd numpy-1.3.0
python setup.py build
python setup.py install --prefix $PKG_ROOT/numpy-1.3.0
cd ..
pushd ../pkg
ln -s numpy-1.3.0 numpy
popd
export PYTHONPATH=$PKG_ROOT/numpy/lib/python2.6/site-packages:$PYTHONPATH
```

- Next 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 ../pkg
ln -s scons-1.2.0 scons
popd
```

- The Boost libraries...:

```
pushd ../pkg
mkdir boost_1_39_0
ln -s boost_1_39_0 boost
popd
cd boost_1_39_0
./bootstrap.sh --with-libraries=python --prefix=$PKG_ROOT/boost
./bjam
./bjam install --prefix=$PKG_ROOT/boost --libdir=$PKG_ROOT/boost/lib
export LD_LIBRARY_PATH=$PKG_ROOT/boost/lib:$LD_LIBRARY_PATH
cd ..
pushd ../pkg/boost/lib/
ln -s libboost_python*-1_39.dylib libboost_python.dylib
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
make install

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

- Finally matplotlib:

```
cd matplotlib-0.98.5.3
python setup.py build
python setup.py install --prefix=$PKG_ROOT/matplotlib-0.98.5.3
cd ..
pushd ../pkg
ln -s matplotlib-0.98.5.3 matplotlib
popd
cd ..
```

4.2.1 Compiling escript

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

```
cd escript.d/scons
cp mac_standalone_options_example.py YourMachineName_options.py

echo $PKG_ROOT
```

Edit the options file and put the value of PKG_ROOT between the quotes in the PKG_ROOT= line. For example:

```
PKG_ROOT="/home/bob/stand/pkg"
```

```
cd ../bin
```

Modify the STANDALONE line of escript to read:

STANDALONE=1

Start a new terminal and go to the stand directory.

```
export PATH=$(pwd)/pkg/scons/bin:$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
```

4.3 Additional Functionality

To perform visualizations 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 also need libjpeg and its headers as well as libpng[4] and its headers
- A tool to visualize VTK files - for example Mayavi or LLNL's VisIt.

[4] libpng requires zlib to build