Installation guide for *esys-Escript*

Release - 3.2
(r3422)

Escript development team

December 15, 2010

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 *esys-Escript*[1] on your computer. To learn how to use Escript please see the Cookbook, User's guide or the API documentation. If you use the Debian or Ubuntu packages to install then the documentation will be available in `/usr/share/doc/escript`, otherwise (if you haven't done so already) you can download the documentation bundle from launchpad.

Escript 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 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. If you choose to compile from source your options are to

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

Compiling Escript when its dependencies are already installed is discussed in Chapter 3. To compile Escript 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.

Once everything is installed you can test your installation using the Python scripts in `examples.zip` or `examples.tar.gz`[2]. Unpack the examples and try to run the following from a terminal:

```
run-escript poission.py
```

If this produces a VTK file called `u.vtu` then you are likely to have a functional Escript installation. You can try and visualize the VTK data or delete the file. For visualization we suggest using `VisIt`[3] or `MayaVi`[4] which are both freely available.

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

[1] For the rest of the document we will drop the *esys-*

[2] These should either be in `escript.d/release/doc` or in the case of Debian, in `/usr/share/doc/escript`.

[3] `https://wci.llnl.gov/codes/visit/`

[4] `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
- Windows – Section 2.3.

Note that the binary packages do not support OpenMP[1] or MPI[2]. If you need these features you will need to compile Escript from source (see Section 3.2 and Section 4.1.4.)

2.1 Linux binary installation

Escript 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.

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

If you are using Debian 5.0("Lenny"), Ubuntu 8.10("Intrepid Ibex") or greater, then see Section 2.1.1. For other linux distributions refer to Section 2.1.2.

2.1.1 Debian and Ubuntu

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 the distribution codename (eg "`lenny`" or "`jaunty`") and `A` is the architecture. For example, `escript-3.0-1-lenny_amd64.deb` would be the file for lenny for 64bit processors. To install Escript download the appropriate `.deb` file and execute the following commands as root (you need to be in the directory containing the file):

(For users of Ubuntu 10.10 *"Maverick Meercat"* only)
You will need to either install `aptitude`[3] or replace use `apt-get` where this guide uses `aptitude`.

```
sudo apt-get install aptitude
```

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

Installing escript should not remove any packages from your system. If aptitude suggests removing escript, then choose 'N'. It should then suggest installing some dependencies choose 'Y' here. If it suggests removing escript-noalias then agree.

If you use sudo (for example on Ubuntu) enter the following instead:

[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.
[3] Unless you are short on disk space `aptitude` is recommended

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

This should install Escript 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_3.2_i386.tar.bz2`) and a group of required programs (`escript-support_3.0_i386.tar.bz2`) (Note that the support bundle is version 3.0 not 3.2) . For 64-bit Intel and Amd processors substitute `amd64` for `i386`. This point release uses the same support bundle as previous releases so if you already have it you don't need a new version.

```
tar -xjf escript-support_3.0_i386.tar.bz2
tar -xjf escript_3.2_i386.tar.bz2
```

This will produce a directory called `stand` which contains a stand-alone version of Escript 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/run-escript
```

This should give you a normal python shell. If you wish to save on typing you can add `x/stand/escript.d/bin`[4] to your `PATH` variable (where x is the absolute path to your install).

[4] or whatever you renamed `stand` to.

2.2 MacOS X binary installation

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

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

2.2.1 Stand-alone bundle MacOS X 10.5 ("Leopard")

You will need to download both escript (`escript_3.2_osx.dmg`) and the support files (`escript-support_3.0_osx.dmg`). This point release uses the same support bundle as previous releases so if you already have it you don't need a new version.

- 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[5] and type

```
x/escript.d/bin/run-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).

The previous point release (3.1) installed sucessfully on MacOS X 10.6.2 ("Snow Leopard") but we have not tested this one.

[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.

2.3 Windows binary installation

There is no automated install/uninstall procedure for Escript on Windows at this time.

2.3.1 Dependencies

- Windows XP (this install has not been tested on newer versions).
- (For the MPI version) MPICH2 1.0.8 (http://www.mcs.anl.gov/research/projects/mpich2/)
- pythonxy (http://www.pythonxy.com) or
  - Python 2.5.4 (http://python.org)
  - Numpy 1.3.0 (http://sourceforge.net/projects/numpy/files/NumPy)
- Optional:
  - gmsh 2.4.0 (required to use pycad, must be in your PATH) - http://www.geuz.org/gmsh/
  - matplotlib 0.99 - http://matplotlib.sourceforge.net/

Unpack 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]

[6] Failing to do so my result in the error message: "This application has failed to start because the boost_python-vc71-mt-1_33_1.dll was not found."

CHAPTER
THREE

Building escript from source

This chapter describes how to build Escript 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 is known to compile and run on the following systems:

- Linux using gcc[1]
- Linux using icc on SGI ICE 8200. (At this stage, we do not recommend building with intel-11)[2]
- MacOS X using gcc
- Windows XP using the Visual C compiler (we do not specifically discuss Windows builds in this guide).

If you have compiled a previous version of Escript please note that the format of the `..._options.py` file has changed so you will not be able to reuse old options file for this build.

3.1 External dependencies

The following external packages are required in order to compile and run Escript. 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`.

[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.

[2] There is a bug in icpc-11 related to exception handling and openmp. This results in binaries which crash.

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)
- netpbm (http://netpbm.sourceforge.com)
  - Tools for producing movies from images
- 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.
- Lapack - Available in various versions from various places.
  Currently only used to invert dense square matrices larger than 3x3.

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 into your options file:

- `scons/TEMPLATE_linux.py` (Linux and MacOS X desktop)
- `scons/TEMPLATE_windows.py` (Windows XP)

This options file controls which features and libraries your build of escript will attempt to use. For example to use OpenMP or MPI you will need to enable it here.

If you want to try escript out without customising your build, then change directories to `escript.d` and

```
scons
```

If you are using a relatively recent version of Ubuntu this may produce errors. The problem is the name of a library in Ubuntu packages. Find the line in the options file you just created which reads

```
#boost_libs = ['boost_python-mt']
```

You will need to change the name so it has a python version after it. For example:

```
boost_libs = ['boost_python-mt-py26']
```

Then run scons.

If this works, then you can skip to Section 3.2.3. If not, then you will need to make some more modications to the the file. Read on.

The template files contain all available options with a comment explaining the purpose of each. Check through the file and ensure that the relevant paths and names are correct for your system and that you enable optional components that you wish to use. For example, to use netCDF, find the netcdf-related lines, uncomment them (i.e. remove the # at the beginning of the lines) and change them according to your installation:

```
netcdf = True
netcdf_prefix = '/opt/netcdf4'
netcdf_libs = ['netcdf_c++', 'netcdf']
```

In this example, netCDF *header* files must be located in `/opt/netcdf4/include`[3] and the *libraries* in `/opt/netcdf4/lib`[4]. If this scheme does not apply to your installation then you may also specify the include-path and library-path directly like so:

```
netcdf_prefix = ['/usr/local/include/netcdf', '/usr/local/lib']
```

The order is important: the first element in the list is the *include*-path, the second element is the *library*-path and both must be specified.

If a line in the options file is commented out and you do not require the feature, then it can be ignored.

To actually compile (if you have n processors, then you can use `scons -j`n 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 enabled an optional dependency and the library or include files could not be found you will be notified and the build will stop.

Note, that you can override all settings from the options-file on the scons command line. For example, if you usually build an optimized version but would like to build a debug version into a separate directory without changing your default settings, you can use:

```
scons debug=1 prefix=debugbuild
```

This will install the binaries and libraries built in debug mode into directories underneath `./debugbuild`.

To run the unit test suite that comes with the source code issue

```
scons all_tests
```

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

3.2.1 Compilation with OpenMP

OpenMP is generally enabled by setting compiler and linker switches. For the most common compilers these are automatically set by build system and all you have to do is set the `openmp` option to True in your options file. If this does not work or your compiler is different, then consult your compiler documentation for the precise switches to use and modify the `omp_flags` and `omp_ldflags` variables in your options file. For example, for gcc compilers which support OpenMP use:

[3] or .../include32 or .../include64 or .../inc

[4] or .../lib32 or .../lib64

```
openmp = True
omp_flags = '-fopenmp'
omp_ldflags = '-fopenmp'
```

(The two latter settings can also be left out as this is the default OpenMP on gcc.)

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

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

3.2.2 Compilation with MPI

You need to have MPI preinstalled on your system. There are a number of implementations so we do not provide any specific advice here. Set the following variables in your options file to according to your installation:

- `mpi`
  which MPI implementation (flavour) is used. Valid values are

  `none` MPI is disabled

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

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

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

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

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

- `mpi_prefix`
  where to find MPI headers and libraries (see netCDF example above)

- `mpi_libs`
  which libraries to link to.

To test your build using 6 processes enter:

```
export ESCRIPT_NUM_PROCS=6
scons all_tests
```

and on 2 processes with 4 threads each (provided OpenMP is enabled)[5]:

```
export ESCRIPT_NUM_THREADS=4
export ESCRIPT_NUM_PROCS=2
scons all_tests
```

Alternatively, you can give a hostfile

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

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

[5] Unless your system has 8 cores expect this to be slow

3.2.3 Difficulties

Mismatch of runtime and build libraries

Most external libraries used by Escript are linked dynamically. This can lead to problems if after compiling Escript these libraries are updated. The same applies to the installed Python executable and libraries. Whenever these dependencies change on your system you should recompile Escript to avoid problems at runtime such as load errors or segmentation faults.

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. Section 4.3 lists additional visualisation tools.

4.1 Installing from source for Linux

4.1.1 Dependencies

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`

4.1.2 Preliminaries

You will also need a copy of the Escript 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
```

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

doc

Before you start copy the Escript 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
```

4.1.3 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 ..
```

4.1.4 Compiling escript

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

```
cd escript.d/scons
cp TEMPLATE_linux.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`. If you wish to build with OpenMP, MPI or configure other aspects of the system take a quick look at Section 3.2.

You will need to edit your options file and specify where to find python, boost and netcdf. (replace `x/stand` with the path to `stand`)

```
#python_prefix = '/usr/local'
```

should be

```
python_prefix = ['x/stand/pkg/python/include/python2.6', 'x/stand/pkg/python/lib']
```

```
#boost_prefix = '/usr/local'
```

should be

```
boost_prefix = ['x/stand/pkg/boost/include/boost-1_39', 'x/stand/pkg/boost/lib']
```

```
#boost_libs = ['boost_python-mt']
```

should be

```
boost_libs= ['boost_python-gcc43-mt-1_39']
```

```
#netcdf = True
```

should be

```
netcdf = True
```

```
#netcdf_prefix = '/usr/local'
```

should be

```
netcdf_prefix = ['x/stand/pkg/netcdf/include', 'x/stand/pkg/netcdf/lib']
```

```
cd ../bin
```

Modify the STANDALONE line of `run-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/run-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
```

4.1.5 Cleaning up

Once you are satisfied, the `escript.d/build` and `stand/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 `pkg` directory:
  - `scons`, `scons-1.2.0`
- `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.

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)

Once these tools have been installed, follow the linux instructions in Section 4.1.2. 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.

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`[3] and its headers
- A tool to visualize VTK files - for example Mayavi or LLNL's VisIt.

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