english[2013/08/17]

# **ANUGA** Internal Tools Manual

*Release 1.3.7*

Geoscience Australia and the Australian National University

Tuesday February, 2015, 17$^{th}$minutes to in the afternoon

Geoscience Australia
Email: ole.nielsen@ga.gov.au

**ANUGA**

Manual typeset with LaTeX

**Credits**:

- **ANUGA** was developed and is maintained by Stephen Roberts, Ole Nielsen, Duncan Gray and Jane Sexton.

**License**:

- **ANUGA** is freely available and distributed under the terms of the GNU General Public Licence.

# CONTENTS

CHAPTER

**ONE**

# Internal Tools

## 1.1 Introduction

This document describes the tools written for internal **ANUGA** use at Geoscience Australia.

These tools are necessarily ad-hoc in nature and of possibly limited general use. If a tool becomes useful to a wider audience it may be moved into the 'ANUGA Tools Manual'.

The tools documented below are:

- acceptance_tests
- cmpsww
- event_selection
- mk_digest
- plotcsv
- tar_file
- update_DVD_images
- write_large_files

### 1.1.1 acceptance_tests

This collection of tests is designed to speed up and automate acceptance testing of a 'cluster' of compute servers. The tests are highly dependent on the installed software environment, so may have limited use outside Geoscience Australia, though the system design does lend itself to change.

The suite of tests checks:

- installed software, such as python installed packages
- availability of NFS mounted filesystems
- ability to ssh to each compute node from the master
- various aspects of parallel computation

The tests are a collection of self-contained acceptance 'testlets' that will be usually run from a controlling master program, but may be run individually. This is very useful when developing a new test, as it can be run by itself until correct.

#### 1.1.1.1 Using acceptance_tests

The acceptance tests are designed to be run from the cluster 'master node', so you must `ssh` to that machine. It is assumed the acceptance tests code suite itself has been installed on the node it is being run from and other required code has been installed on all nodes.

Before running the acceptance tests you must prepare some environment variables:

| | |
|---|---|
| PYTHON | Defines the path to the python executable to use for the sub-tests. |
| PYTHONPATH | The path to the **ANUGA** source directory. |
| EQRMPATH | The path to the EQRM source directory. If not set, EQRM is not tested. |

The first sub-test run dumps the testing environment to the screen as a check.

To run the acceptance tests, do the following:

```
export PYTHON=python2.5 # we want to run python 2.5 in the tests
export PYTHONPATH=/home/r-w/sandpit/ga/anuga_core/source/
# EQRMPATH not set
python test_all.py
```

While the tests are running, you will see the results of each test listed to the screen. Don't worry about catching this output; everything is written to a log file `anuga.log`.

### 1.1.2 cmpsww

The `cmpsww` program is used to compare two SWW files for some approximation of *equality*. The user must be able to define what to compare in the two files, as well as set tolerances for 'how close is close'.

#### 1.1.2.1 Using cmpsww

The usage printed by the program is:

```
Usage: cmpsww.py <options> <file1> <file2>
where <options> is zero or more of:
                  -h        print this help
                  -a <val>  set absolute threshold of 'equivalent'
                  -r <val>  set relative threshold of 'equivalent'
                  -g <arg>  check only global attributes specified
                            <arg> has the form <globname>[,<globname>[,...]]
                  -t <arg>  check only timesteps specified
                            <arg> has the form <starttime>[,<stoptime>[,<step>]]
                  -v <arg>  check only the named variables
                            <arg> has the form <varname>[,<varname>[,...]]
and <file1> and <file2> are two SWW files to compare.

The program exit status is one of:
   0    the two files are equivalent
   else the files are not equivalent.
```

Note that if no globals, variable or timesteps are specified, the program checks all globals and all variables for all timesteps.

#### 1.1.2.2 Bugs

The `cmpsww` program is still being developed and needs to change in concert with the methodology of determining if an SWW file is as expected.

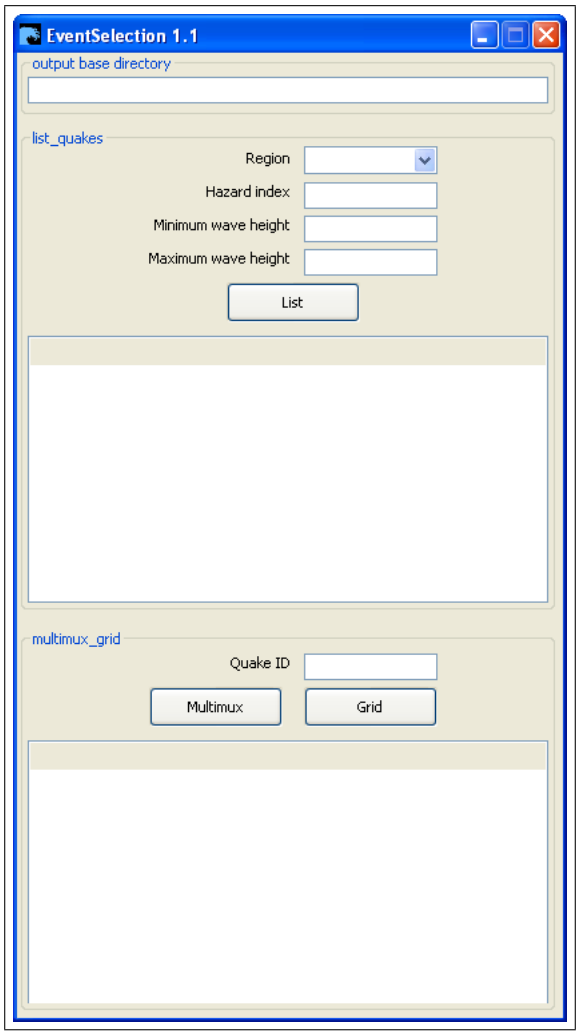
### 1.1.3 event_selection

`event_selection` is a graphical program used to select earthquake events.

It designed to run under both Windows and Linux.

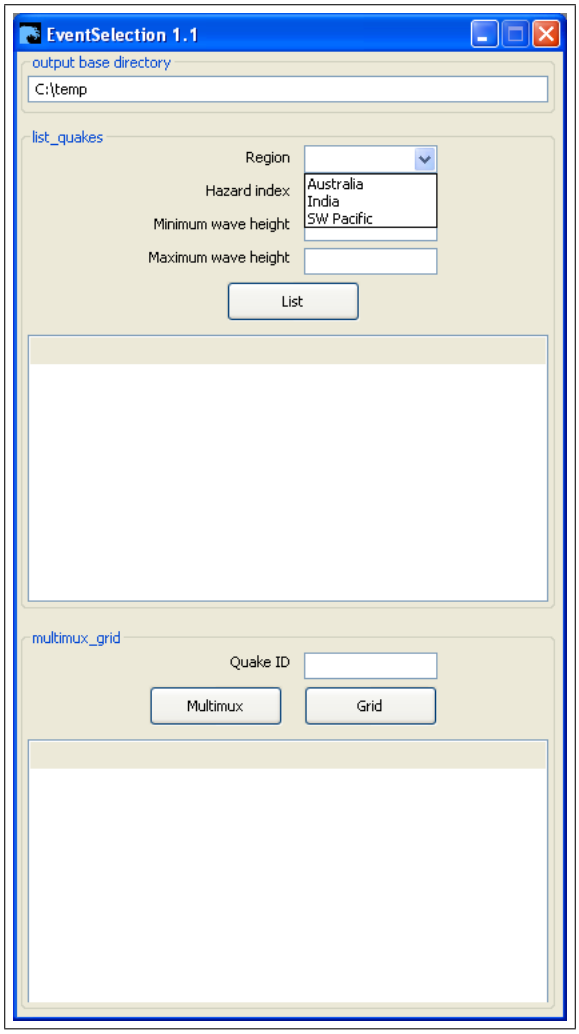
#### 1.1.3.1 Using event_selection

Once you start the `event_selection` program you will see:



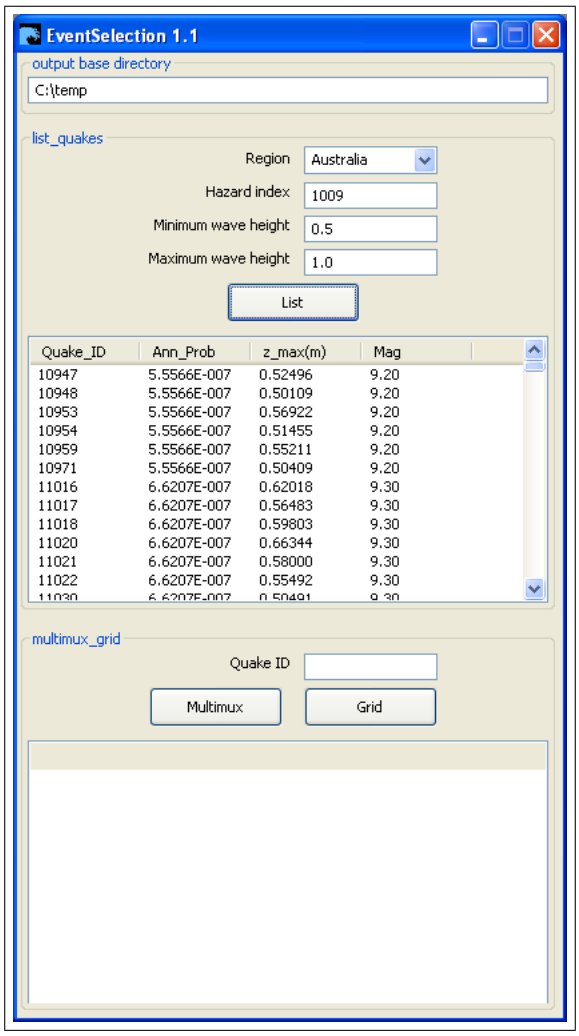

Before using the program, you need to set the *output base directory* field at the top of the window. The program needs to write some data files and this field tells the program where to write them. Just click in the box to select a directory somewhere in your filesystem.

We set the directory to `C:\temp`. Next, you need to select the *Region* from the drop-down list:

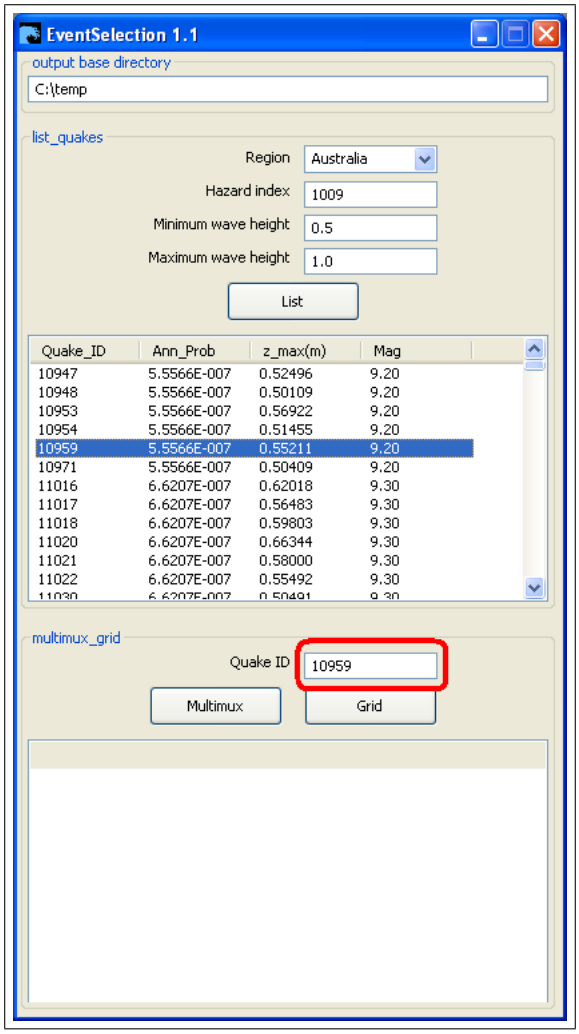



At this point you have set the values that will probably never change. If you close the program at this point, then the two values set (*base directory* and *Region*) and the three fields below (*Hazard index*, *Minimum wave height* and *Maximum wave height*) will be remembered and restored the next time you run the program. This data is stored in a file `event_selection.cfg` in the `event_selection` install directory.

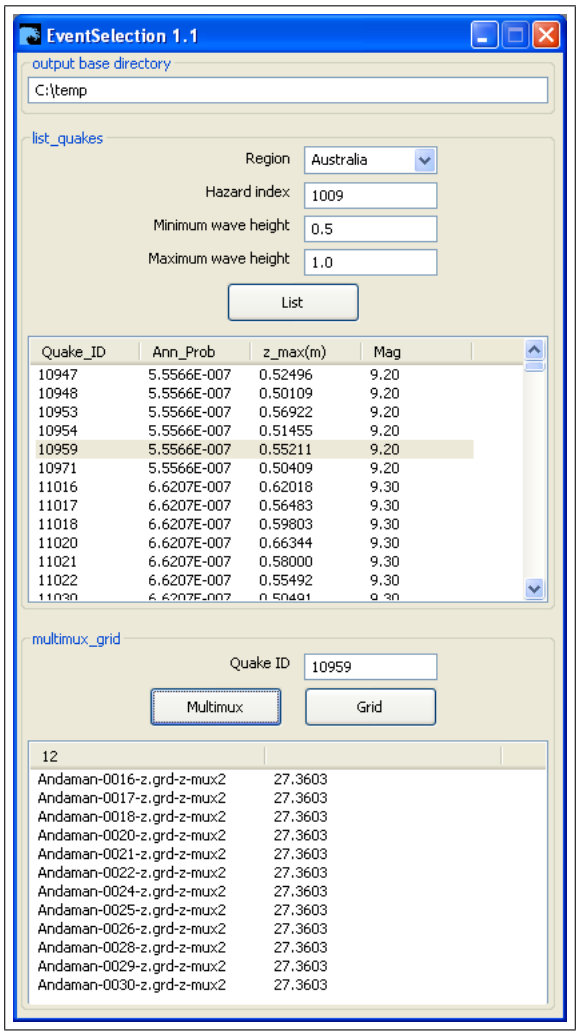
Now you need to enter data specific to a particular event you are going to model. Fill in the *Hazard index* (location in the database of the point where the hazard is measured), *Minimum wave height* and *Maximum wave height* values and click on the *List* button:



The program has filled the text box below the *List* button with events that satisfy your listed requirements. You need to select one of these events, which puts the *Quake_ID* number into the *Quake_ID* textbox below:



Now you can click on either the *Multimux* or *Grid* buttons. Clicking on the *Multimux* button gives us:



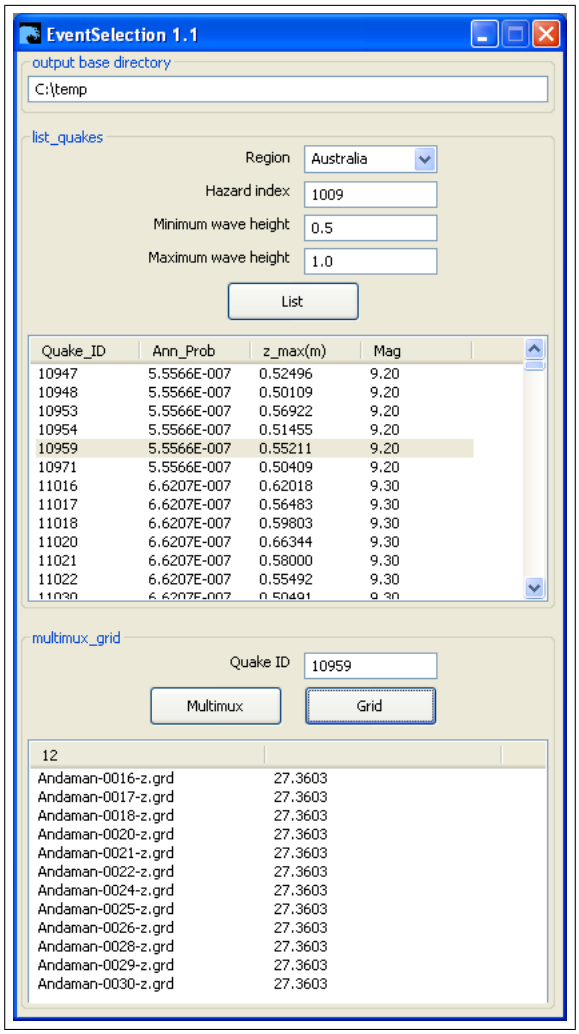

If you now look in the output directory `C:\temp` you will see that two directories have been created:

```
10959
Results_Australia_1009_0.50_1.00
```

The `Results_Australia_1009_0.50_1.00` directory contains the `fault.xy` and `quake_prob.txt` files used during the calculation of the multimux results. The `Results` directory name contains the region name, hazard index and minimum and maximum wave heights in an encoded form.

The `10959` directory contains the multimux data for the selected Quake_ID in a file called `event_list`.

The *Grid* button was installed to allow the selection of seafloor deformation grid data. Clicking on this button shows:




and writes some extra files into the `Results_Australia_1009_0.50_1.00` directory:

```
event_010959.list
faults_010959.params
```

The `event_010959.list` file contains:

```
12
Andaman-0016-z.grd-z-mux2 27.3603
Andaman-0017-z.grd-z-mux2 27.3603
Andaman-0018-z.grd-z-mux2 27.3603
Andaman-0020-z.grd-z-mux2 27.3603
Andaman-0021-z.grd-z-mux2 27.3603
Andaman-0022-z.grd-z-mux2 27.3603
Andaman-0024-z.grd-z-mux2 27.3603
Andaman-0025-z.grd-z-mux2 27.3603
Andaman-0026-z.grd-z-mux2 27.3603
Andaman-0028-z.grd-z-mux2 27.3603
Andaman-0029-z.grd-z-mux2 27.3603
Andaman-0030-z.grd-z-mux2 27.3603
```

The `faults_010959.params` file contains:

```
GENERATED=Tue Jun 09 17:36:28 2009
REGION=Australia
HAZARD_INDEX=1009
WAVE_HEIGHTS=[0.50,1.00]
QUAKE_ID=10959
WEIGHT_FACTOR=27.360300
DATA_PATH=N:\georisk_models\tsunami\models\PTHA\Australia\static_2m
GRID=Andaman\Andaman-0016-z.grd
GRID=Andaman\Andaman-0017-z.grd
GRID=Andaman\Andaman-0018-z.grd
GRID=Andaman\Andaman-0020-z.grd
GRID=Andaman\Andaman-0021-z.grd
GRID=Andaman\Andaman-0022-z.grd
GRID=Andaman\Andaman-0024-z.grd
GRID=Andaman\Andaman-0025-z.grd
GRID=Andaman\Andaman-0026-z.grd
GRID=Andaman\Andaman-0028-z.grd
GRID=Andaman\Andaman-0029-z.grd
GRID=Andaman\Andaman-0030-z.grd
FAULT=Andaman
```

### 1.1.4 Installing event_selection

There is an installer program used to install `event_selection` on a Windows machine (usually found at `georisk\downloads\event_selection`). The installer is generated by moving into the `installer` directory and right-clicking on the `EventSelection.nsi` file and selecting `Compile NSIS script`. You must have installed the NSIS package for this to work. Get it from http://nsis.sourceforge.net/Main_Page.

Once you have installed `event_selection` on your Windows machine you will have a desktop icon and Start menu entry to start the program with.

Under Linux just execute the `event_selection.py` program, either from the console or from a desktop icon or menu entry you created.

#### 1.1.4.1 Requirements

Various pieces of python software must be installed before `event_selection` can be used. These are:

- wxpython - a python package
- NSIS - a Windows installer generator (required if creating a Windows installer)

#### 1.1.4.2 Bugs

The look of `event_selection` under Linux is wrong – it needs to be rewritten using sizers for GUI layout.

### 1.1.5 mk_digest

`mk_digest.py` is a small program used to create an MD5 digest of a file. The digest string is written into a file.

This program is used in the Patong Beach validation file refresh process.

#### 1.1.5.1 Using mk_digest

```
usage: mk_digest.py <datafile> <digestfile>
where <datafile>   is the file for which we create a digest string
      <digestfile> is the created file that contains the hex string.
```

#### 1.1.5.2 Installing mk_digest

Installation is not required, just run the program.

### 1.1.6 plotcsv

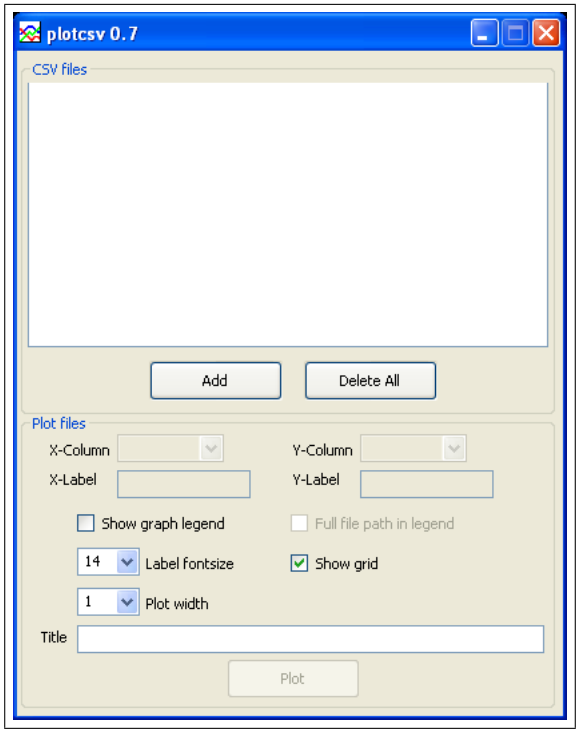
`plotcsv` is a GUI program to quickly plot selected columns of one or more CSV files onto a graph screen. Once the desired graph is plotted you may save the plot as a picture file.

The program is designed to run under both Windows and Linux.

The CSV files used *must* have column header information as the first line as the column header values are used during the plotting process.

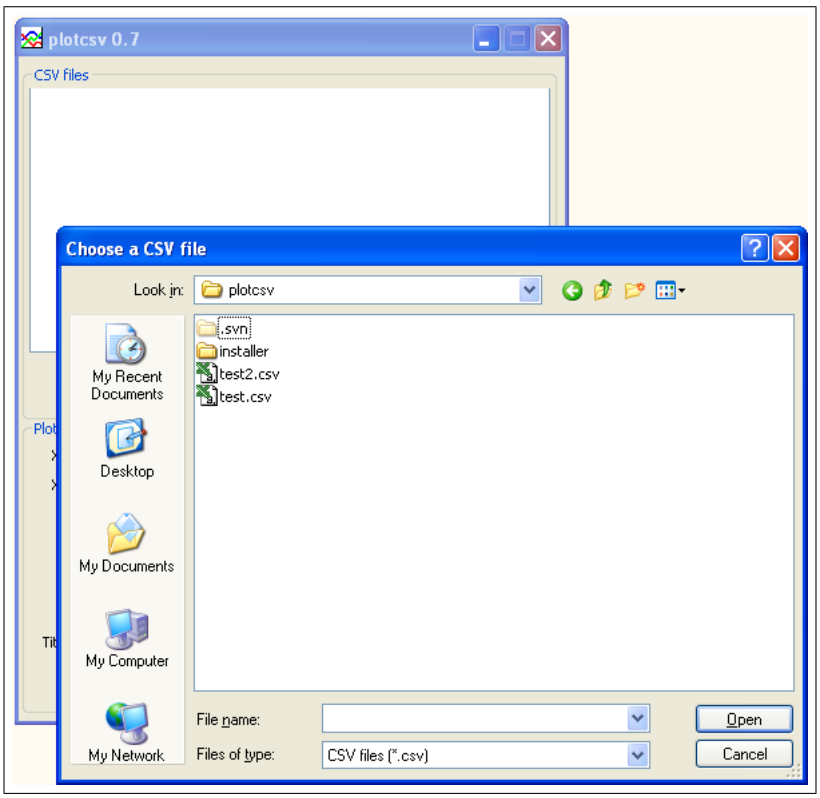
#### 1.1.6.1 Using plotcsv

Start the program by selecting it from the Start menu or double-clicking on the desktop icon. You will see the following window:

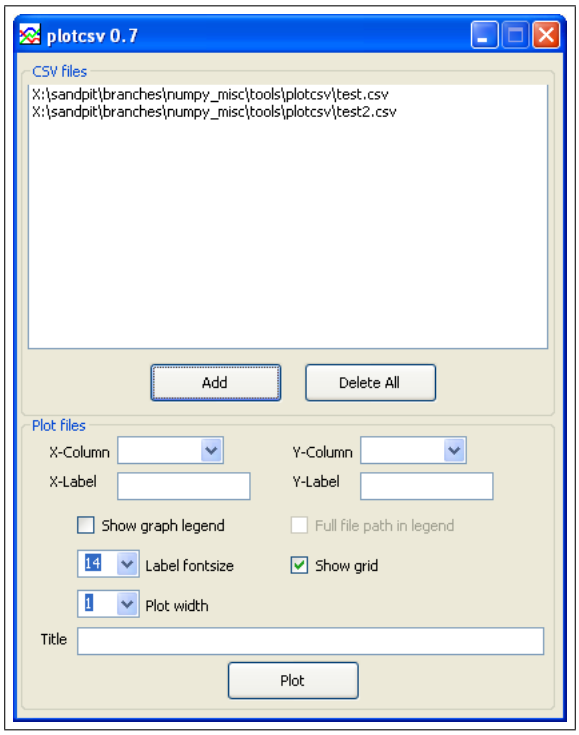



The first thing you must do is select one or more CSV files to plot. The files you are going to plot are listed in the textbox at the top of the screen. There is nothing there because this is the first time you have run `plotcsv`. Note that `plotcsv` will remember the selected files, as well as other information, when you next start the program. This files is `plotcsv.cfg` and it is stored in the `plotcsv` install directory.

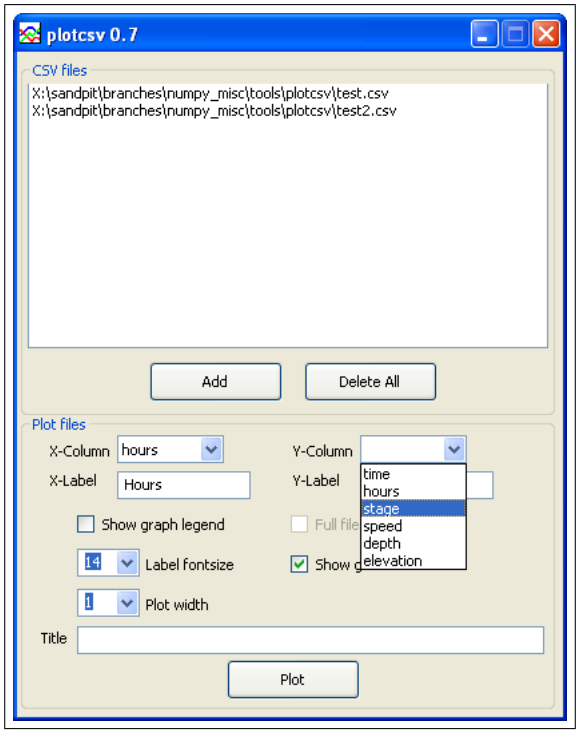
This screen shot shows what happens when you click on the *Add* button - you get a file selector that lets you choose the CSV files to plot.



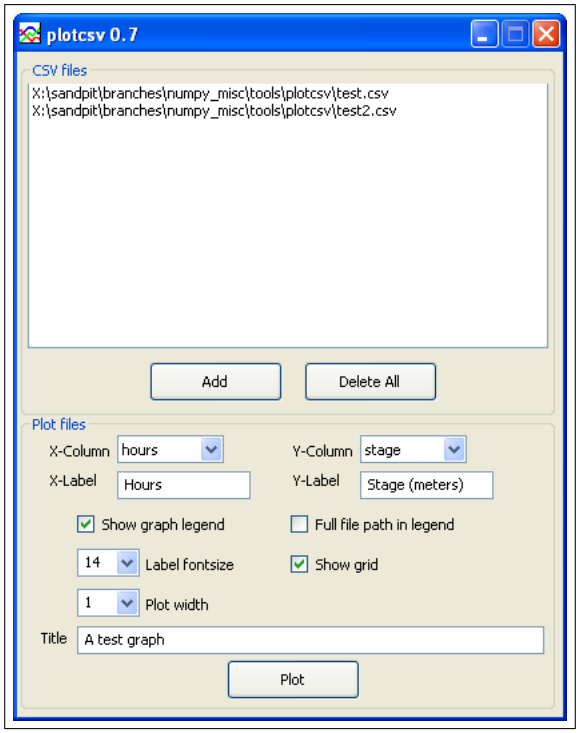
In this example we selected both `test.csv` and `test2.csv`.



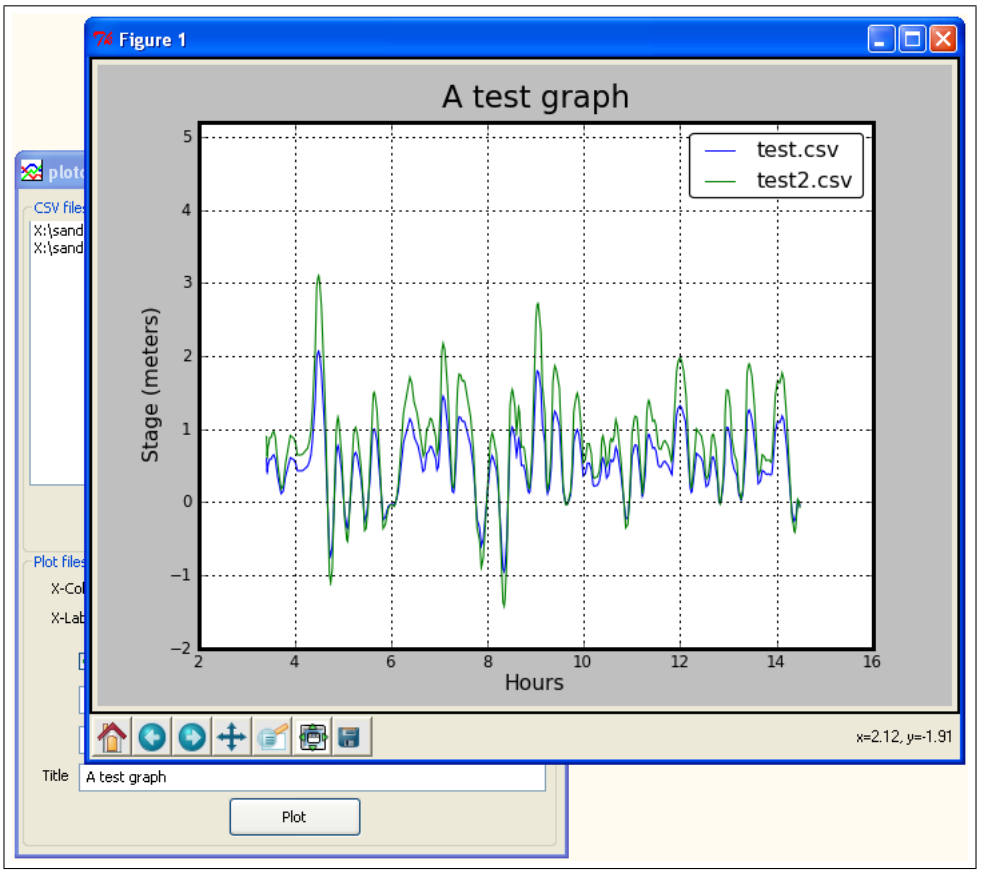
You must now set the column data to display the X and Y axis of your plot. The *X-Column* and *Y-Column* listboxes are used to set which column data to use. In this example we are going to plot Stage versus Time, so we select the appropriate columns below:



Note that choosing a column to plot also sets the text in the *X-Label* and *Y-Label* textboxes. You can change this text and, in this example, we want to change the stage axis text to *Stage (meters)*. We also add some title text and turn on the graph legend:



Finally, once we are ready, we click on the *Plot* button and see our plot:




You are free to configure the plot, make it larger, save a picture file, etc. Closing the plot window shuts down the application (see Bugs section below).

#### 1.1.6.2 Installing plotcsv

For Windows execute the `plotcsv_X.X.exe` file in `N:\georisk\downloads\plotcsv`. This will install `plotcsv` into your `C:\Program Files` directory and create a desktop icon.

Linux needs no installation, just run the program.

#### 1.1.6.3 Building plotcsv for Windows

The source directory for `plotcsv` contains an `installer` directory. Just right-click on the `plotcsv.nsi` file and select "Compile NSIS Script". You must ihave the NSIS installer installed, of course. Get it from http://nsis.sourceforge.net/Main_Page.

#### 1.1.6.4 Bugs

The mixture of matplotlib and wxpython isn't successful - you only get one plot and then you must close the application. Using the `wx_mpl_bars.py` example from http://eli.thegreenplace.net/2008/08/01/matplotlib-with-wxpython-guis/, rewrite `plotcsv` to have the parameter changes (such as title text) show up immediately in the current plot.

The look of `plotcsv` under Linux is wrong – it needs to be rewritten using sizers for GUI layout.

### 1.1.7 tar_file

The `tar_file.py` program is used to tar and compress a file or directory into a *.tgz file. We have a python function to do this as we can't use a local *tar* program, as this wouldn't work under Windows.

The associated `untar_file.py` program reverses the above process.

These two programs are used in the Patong Beach validation suite.

#### 1.1.7.1 Using tar_file

```
tar_file.py <tarfile> <file1> [<file2>, ...]
```

where *tarfile* is the path to the tar file to create, and *file?* is the path to a file or directory to include.

#### 1.1.7.2 Using untar_file

```
untar_file.py <tarfile> [<output_directory>]
```

where *tarfile* is the path to the file to untar, and *output_directory* is the directory to write the results into.

If *output_directory* is not specified then the compressed file is unpacked into the current directory.

#### 1.1.7.3 Installing tar_file

No installation is required, just run the program.

### 1.1.8 update_DVD_images

`update_DVD_images` is a program used to create the DVD image filesystems that were burnt to DVD for the 2009 East Coast Tsunami Inundation study.

#### 1.1.8.1 Using update_DVD_images

To use the `update_DVD_images` program, just execute the program:

```
python update_DVD_images.py <name of jurisdiction>
```

Currently, the jurisdiction names are:

- BatemansBay
- GoldCoast
- Gosford
- Hobart

So to recreate the GoldCoast DVD image sub_directory, do:

```
python update_DVD_images.py goldcoast
```

Note that the case of the jurisdiction name doesn't matter.

The program will create a new sub-directory with the *formal* jurisdiction name (see below) in the current directory. The old jurisdiction sub-directory is deleted first.

#### 1.1.8.2 Configuration

Here we discuss how to configure `update_DVD_images` to handle a new jurisdiction or change what files/directories are copied.

In `update_DVD_images.py` there are a set of dictionaries that control what is done for each jurisdiction.

The first dictionary is `source_jurisdiction_path` which maps the lowercase jurisdiction name to the dictionary defining that particular jurisdiction:

```
source_jurisdiction_path = {'hobart': hobart_data,
                            'batemansbay': batemans_bay_data,
                            'gosford': gosford_data,
                            'goldcoast': gold_coast_data}
```

If you create a new jurisdiction, you need to add another line to the above dictionary.

In the case of the GoldCoast jurisdiction, we see that the dictionary for the GoldCoast is `gold_coast_data`:

```
gold_coast_data = \
{'jurisdiction':    'GoldCoast',        # jurisdiction name

 # paths to various source directories
 'data_src_path':   'data/queensland/gold_coast_tsunami_scenario_2009/anuga',
 'arcgis_src_path': 'data/queensland/gold_coast_tsunami_scenario_2009/ArcGIS',
 'proj_src_path':   'sandpits/lfountain/anuga_work/production/gold_coast_2009',

 # paths to destination directories (under 'jurisdiction' root)
 'data_dst_path':   'anuga',
 'proj_dst_path':   'project',
 'arcgis_dst_path': 'ArcGIS',

 # copy or create whole directories
 'make_dst_dirs':   ['outputs'],
 'copy_data_dirs':  ['boundaries'],

 # copy 'data' files or directories
 'copy_data_files': ['outputs/Event1_HAT', 'outputs/Event1_MSL',
                     'outputs/Event2_HAT', 'outputs/Event2_MSL',
                     'outputs/Event3_HAT', 'outputs/Event3_MSL'
                    ],

 # copy 'project' files or directories
 'copy_proj_files': ['build_elevation.py', 'export_results_max.py',
                     'get_runup.py', 'project.py', 'run_model.py',
                     'setup_model.py', 'build_urs_boundary.py',
                     'combine_gauges.py', 'get_timeseries.py',
                     'run_multiple_events.py'
                    ],

 # copy 'arcgis' files or directories
 'copy_arc_files':  ['MainBeach.mxd', 'SurfersParadise.mxd', 'GoldCoast.mxd',
                     'PalmBeach.mxd', 'Collangatta.mxd'
                    ]
}
```

The first key is `jurisdiction`, which maps to a string defining the jurisdiction formal name. This name is used to create the output DVD staging directory.

```
'jurisdiction':    'GoldCoast',
```

The next three key values define the complete paths to source directories in the production filesystem:

```
# paths to various source directories
'data_src_path':   'data/queensland/gold_coast_tsunami_scenario_2009/anuga',
'arcgis_src_path': 'data/queensland/gold_coast_tsunami_scenario_2009/ArcGIS',
'proj_src_path':   'sandpits/lfountain/anuga_work/production/gold_coast_2009',
```

These key values are used along with a master path variable defined earlier in `update_DVD_images.py` to create the complete paths to source directories:

```
main_path = '/nas/gemd/georisk_models/inundation'
```

For example, the full path to the 'data' source directory would be:

```
data_src_path = os.path.join(main_path, j_dict['data_src_path'])
```

where `j_dict` would be a reference to the jurisdiction dictionary controlling the process (`gold_coast_data` in this case).

The next three definitions define the names of output directories in the staging directory:

```
# paths to destination directories (under 'jurisdiction' root)
'data_dst_path':   'anuga',
'proj_dst_path':   'project',
'arcgis_dst_path': 'ArcGIS',
```

These three names are combined with the current directory and the jurisdiction staging directory name to produce the full path to output directories:

```
data_dst_path = os.path.join(os.getcwd(), j_name, j_dict['data_dst_path'])
proj_dst_path = os.path.join(os.getcwd(), j_name, j_dict['proj_dst_path'])
arcgis_dst_path = os.path.join(os.getcwd(), j_name, j_dict['arcgis_dst_path'])
```

Note that `j_name` is the jurisdiction name. So in this case, we would create the output directories:

```
./GoldCoast/anuga               # data directory
./GoldCoast/project             # project files directory
./GoldCoast/ArcGIS              # ArcGIS files
```

The next two key values define the names of empty directories to create or names of complete directories to copy to the `data_dst_path` directory:

```
# copy or create whole directories
'make_dst_dirs':   ['outputs'],
'copy_data_dirs':  ['boundaries'],
```

The values here are lists of one or more directories to create or copy. If there are no directories to create/copy, just use an empty list.

Next, we define which individual files we copy to the destination data directory:

```
# copy 'data' files or directories
'copy_data_files': ['outputs/Event1_HAT', 'outputs/Event1_MSL',
                    'outputs/Event2_HAT', 'outputs/Event2_MSL',
                    'outputs/Event3_HAT', 'outputs/Event3_MSL'],
```

Again we have a list of files to copy. Note that we must specify the path following the `data_dst_path` variable (`anuga` in this case), so we specify the directory under `anuga` and then the source file (or directory). Also note that we can copy a simple file or complete directory here.

You *must* create each target directory as an empty directory before copying files. That is why `outputs` appears in the `make_dst_dirs` key-value definition above.

Similarly, we now define 'project' files to copy:

```
 # copy 'project' files or directories
 'copy_proj_files': ['build_elevation.py', 'export_results_max.py',
                     'get_runup.py', 'project.py', 'run_model.py',
                     'setup_model.py', 'build_urs_boundary.py',
                     'combine_gauges.py', 'get_timeseries.py',
                     'run_multiple_events.py'],
```

These files (or directories) will be copied from the path defined in the `proj_src_path` variable to the path defined in the `proj_dst_path` variable.

Finally, we define 'arcgis' files or directories to copy:

```
# copy 'arcgis' files or directories
'copy_arc_files':  ['MainBeach.mxd', 'SurfersParadise.mxd', 'GoldCoast.mxd',
                    'PalmBeach.mxd', 'Collangatta.mxd']
```

These files (or directories) will be copied from the path defined in the `arcgis_src_path` variable to the path defined in the `arcgis_dst_path` variable.

#### 1.1.8.3 extra_files

In the same directory as `update_DVD_images` there must be a directory `extra_files`. This directory contains 'scaffolding' files that must exist on the DVD as well as jurisdiction-specific files that may be modifications of project files that replace those files on the DVD.

All files in the `extra_files` directory are copied to each jurisdiction DVD staging directory. All top-level directories that *aren't* named for a jurisdiction are also copied to each staging directory.

Each directory named for a jurisdiction will be copied to the staging directory if the directory has the same name as the jurisdiction staging directory we are creating. This jurisdiction directory would normally contain jurisdiction-specific scaffolding files, such as `index.html`, etc, as well as modified project files.

### 1.1.9 update_lic_checksum

The `update_lic_checksum` program is used to update all licence files (`*.lic`) in a filesystem sub_tree.

The `create_lic_file` program is used to create a licence file that controls one or more data files.

#### 1.1.9.1 Using update_lic_checksum

The program is used:

```
update_lic_checksum.py [-m <lic_mask>] <directory>
```

where *directory* is the path to the sub_directory containing licence files to update. Normally, `update_lic_checksum` would search for and update all `*.lic` files. If you want to update licence files that have a filename form of `*.txt` then use the `-m *.txt` option.

Note that the licence files being updated must contain well-formed XML data.

#### 1.1.9.2 Using create_lic_file

`create_lic_file` is a program used to create licence files from scratch. It is used so:

```
usage: create_lic_file.py <options> <lic_file> [<filename> ...]
where <options>  is zero or more of:
                  --author <name>
                  -w <name>             - name of the author
                  --publishable [Yes|No]
                  -p [Yes|No]           - is document publishable
                  --accountable <name>
                  -a <name>             - name of person accountable for file
                  --source <string>
                  -s <string>           - source of controlled file
                  --owner <name>
                  -o <name>             - IP owner name
                  --info <string>
                  -i <string>           - IP extra information
      <lic_file> is the name of the licence file to create.
      <filename> is one or more files to control.
```

If the file to be created (*lic_file*) already exists, the program aborts; it will not overwrite any existing file.

You must use the options to specify author name, etc. If these are not overridden the generated licence file will contain default values. For example, if you did this:

```
python create_lic_file.py test.lic README
```

then the output file `test.lic` would contain:

```
<?xml version='1.0' encoding='iso-8859-1'?>
<ga_license_file>
  <metadata>
    <author>rwilson</author>
  </metadata>
  <datafile>
    <filename>README</filename>
    <checksum>1387779554</checksum>
    <publishable>Y</publishable>
    <accountable>rwilson</accountable>
    <source>Generated by ANUGA development team</source>
    <IP_owner>Geoscience Australia</IP_owner>
    <IP_info>For use with the ANUGA test suite</IP_info>
  </datafile>
</ga_license_file>
```

In particular, the *author* and *accountable* values are defaulted with the username from the environment.

Note the default values for these fields:

```
<publishable>Y</publishable>
<source>Generated by ANUGA development team</source>
<IP_owner>Geoscience Australia</IP_owner>
<IP_info>For use with the ANUGA test suite</IP_info>
```

### 1.1.10 write_large_files

This program is actually a suite of programs used to exercise the NetCDF file I/O code.

The NetCDF I/O model has three models of the way data is written:

- The 'classic' model
- The 'large' model
- The 'NetCDF4' model

The *classic* model is usually dismissed as the '2GiB limit' model, but this is an over-simplification. Chunks of data wriiten to a file in one write will contain an offset to the next related chunk 'upstream' in the file. This offset has a 2GiB limit, hence the '2Gib' oversimplification.

The *long* model relaxes some of the limits in the *classic* model.

The *NetCDF4* model allows much larger datasets to be written to a file, along with compression, more than one unlimited dimension, etc.

Some effort is made to simulate the way an **ANUGA** program would write data. In particular, the variables written are interleaved in the way **ANUGA** would write them.

Also, each data value written to the file is a floating point number which encodes variable number, variable 'slice' and index into each slice. This is to ensure that each variable value written is unique and to allow for checking that what we read is what we wrote.

#### 1.1.10.1 Using write_large_files

There are three programs in the `write_large_files` suite:

| | |
|---|---|
| rwi_big_file.py | writes using the *classic* model |
| rwil_big_file.py | writes using the *large* model |
| rwi4_big_file.py | writes using the *NetCDF4* model |

Each of the three programs is used in the same way:

```
Usage: write_large_files <opts> <varsize> [<numvars>]

where <varsize> is a number followed by an optional modifier:
                    1024M or 4G
                the assumed modifier if none is given is 'M'.
  and <numvars> is the number of variables of the above size
                    to write.  If not supplied, 1 is assumed.
                    There can be at most 100 variables.
  and <opts>    is zero or more of:
                    -c s    close & open the output file after
                            each variable slice is read/written,
                    -t rf   time the complete file read,
                    -t wf   time the complete file write,
```

For instance, if we wanted to write a 3GiB 'large' file containing 6 variables we would do:

```
python rwil_big_file.py 512M 6
```

#### 1.1.10.2 Installing write_large_files

No installation is necessary, just execute the programs.

#### 1.1.10.3 Bugs

Instead of having three files, one to test each NetCDF model, just add an extra option to a single program:

```
-c    classic model (default)
-l    large model
-4    NetCDF4 model
```