PyShottab Documentation

Introduction:

pyShottab is a Shot Table creator written in Python (www.python.org). It has two components:

A GUI options editor (shotGui)
A command line shotab creation program (shotCreate)

The GUI editor is written in wxWindows, a portable windowing toolkit, and works on Unix, Windows, and Mac (X and Classic). The command line program is portable to all platforms Python supports (virtually all modern OS's).

Installation:

Under Windows, there are two options. 1) run the program from source (recommended) or 2) simply run the self-installing executable file pyShottab.exe. One can get this file from the pyShottab cdrom, or off of the Seismic group wiki http://seismic.ocean.dal.ca. This installs shotCreate.exe and shotGui.exe in c:\shottab (or wherever was selected).

Under Linux (or Windows running from source) one needs to install the following packages (easy point and click installers for windows, several different formats for Unix/Linux):

- Python 2.2 or greater (python.org)
- wxPython 2.4 unicode (wxPython.org)

Then one can simply run the code as ./shotGui.py or ./shotCreate.py. On Fox.ocean.dal.ca shotGui and shotCreate are symlinks to the .py files that reside in /usr/local/bin, thus one can run "shotGui" instead. To install on a Linux macine with the above packages installed, one can copy the /data/dave/pyShottab_latest directory, and run ./shotGui.py, ./shotCreate.py as well.

Under Solaris the installation requires the above packages and all their dependencies (GTK, Glib, etc). This is an intensive process on an older machine.

Usage:

One passes parameters to this program through an input file. One can create and edit this with shotGui, and perform the conversion with shotCreate. For shotGui the file name is optional, if not given it will use defaults. This file is similar to a windows . ini file and is easy to edit in any text editor if need be.

Unix/Linux:

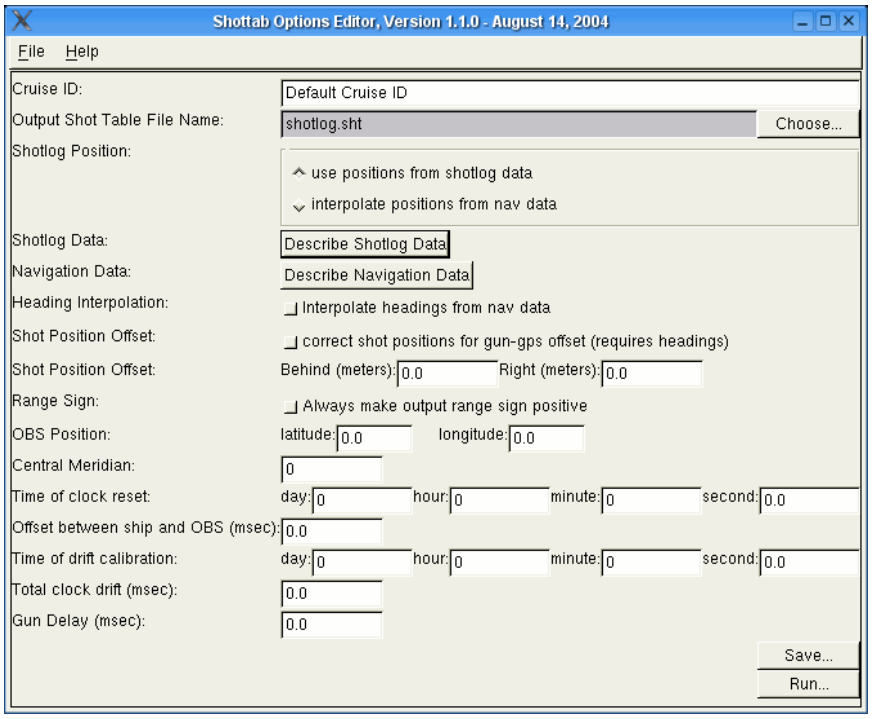
shotGui [file.cfg]
shotCreate file.cfg

Windows:

c:\pyshottab\shotGui [file.cfg]
c:\pyshottab\shotCreate file.cfg

GUI:

The main shotGui dialog is shown below:

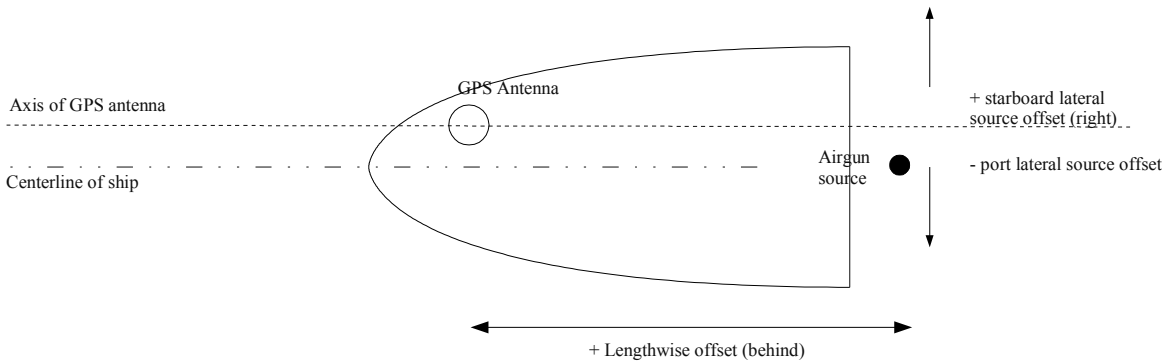


IMPORTANT NOTES:

Shotlog Position - choose to use the lat/long positions from (a) the shotlog, or (b) the ship's navigation log. Use caution if using the shotlog positions. It does not use differential GPS, and therefore may be less accurate. Also, remember to set the "position mode" of the Odetics box to "dynamic" prior to shooting, otherwise the positions will not be correct.

Heading Interpolation - if checked, this interpolates headings from navigation data given their lat/long positions using a running sum algorithm. The first and last few lines are thrown away with this process. If this is not checked, "heading" must be present in the Navigation Data input expression (see below). If position coordinates only exist in the shotlog (ie: nav doesn't exist), the shotlog can be read as navigation data thus allowing correction for gun offset.

Shot Position Offset - the offset of the airgun behind the GPS antenna (positive value), and lateral offset between the ship's GPS antenna and the airguns. If the airguns are to the left (port) of the GPS antenna axis, use a negative sign. If the source is to the right (starboard) of the antenna axis, use a positive value (see figure).



Range Sign – If the range sign is going to be manually calculated, use this so all ranges have a positive sign, making manual calculation easier.

OBS Position – the position of the OBS (decimal degrees). If OBS has not been relocated, this is the drop position, else it is the relocated position.

Central Meridian – the reference longitude for this study area – usually in the center of the area. An integer value equally divisible by 3.

Time of Clock Reset – the time the OBS clock was turned on or reset (GMT). Assumed to be the time of initial calibration.

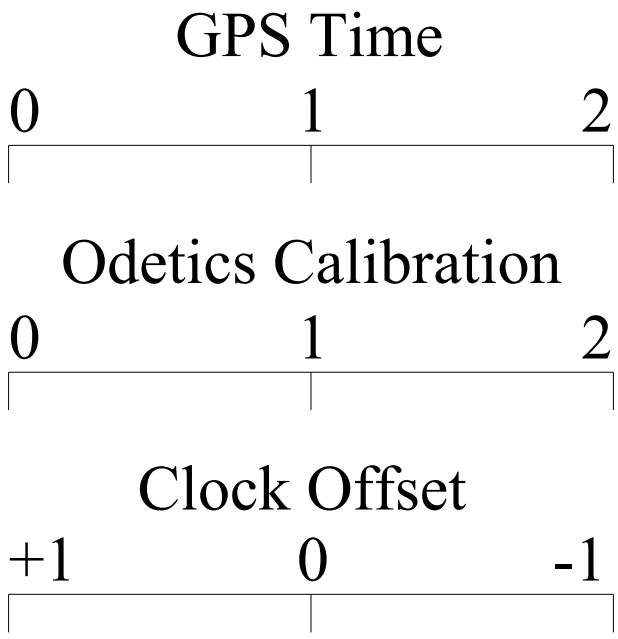
Offset between ship and OBS – the initial offset between the ship's GPS clock and the OBS clock in <u>milliseconds</u>, prior to deployment. This follows the same convention as the TOTAL CLOCK DRIFT (below).

Time of Drift Calibration – the time that the "post deployment offset" for the OBS was determined (GMT).

Total Clock Drift – the total drift in <u>milliseconds</u> of the OBS clock relative to the GPS clock, between the time of deployment and the time of recovery (see figure). The Odetics clock calibration is given in terms of GPS time (seconds). The notation for GPS clock calibration is the following: 1.000 s equals no actual offset, 0.9000 s equals +0.1 s actual offset, and 1.1000 s equals -0.1 s actual offset (subtract the calibration value from 1.0 s to obtain the offset).

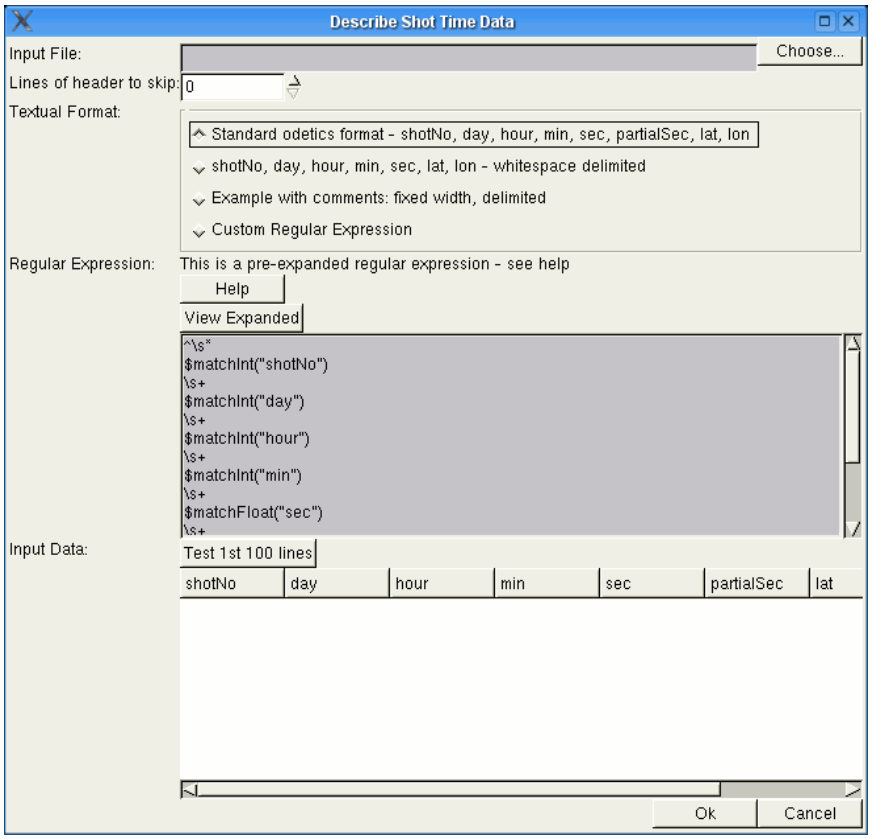
For example, the pre-deployment and post deployment calibrations from the Odetics box are given as 0.999134 s and 1.000512 s respectively. Thus the corresponding offsets are 0.0008660 s, and -0.000512 s respectively. The difference between these offsets is the "total clock drift", in this case -0.001378 s or -1.378 ms (post-deployment minus pre-deployment offset).

Note that the CPU clock calibration is usually 1.8 ms greater than the Odetics clock calibration, therefore 1.8 seconds must be subtracted from it to obtain the GPS equivalent.



Gun Delay – the time delay from when the airguns are signaled to shoot (GPS time), and when they actually shoot - is a static time offset in <u>milliseconds</u> (positive).

Shot Time Format Dialog (Describe Shot Data):



This dialog (similar also to that for parsing the Navigation data) allows one to generically specify the format of a text file which one is to use in the shottab program.

The fields for shot time data are:

- shotNo
- day
- hour
- min
- sec
- partialSec (optional – will use 0 if not included in pattern)
- lat
- lon

Note that if the user selected “interpolate positions from nav data” then the lat and long pattern lines need not be present.

The fields for navigation data are:

- day
- hour
- min
- sec
- lat
- lon

- heading

Note that if the user selected “interpolate headings” they don't need the “heading” line. Conversely, if one wishes to correct for gun offset without interpolating headings, heading data must exist in the nav file.

The “textual format” box allows one to select from some predefined format expressions. Clicking on the “test 1st 100 lines” button below will allow one to test the input to ensure all fields are being converted appropriately. There is an example with comments which shows how to read in several different styles of data (fixed width, space delimited, etc).

The filter expression is a special kind of regular expression (a facility for matching sequences of input – used by Python, Perl, Awk, etc). We are using an augmented regular expression style with the following characteristics:

- It is “verbose” which means it can span multiple lines, and have whitespace separating patterns. It can also have comments (#comment).
- Predefined patterns are available to ease the matching process – these are access as $matchInt(“intName”), $matchFloat(“floatName”), etc.

The $functions() must be “expanded” - that is expanded into a regular expression that matches for an integer, float, etc. Pressing the “View Expanded” button allows one to view the final output. Expanding happens transparently, this button is for utility only.

The “help” button gives some assistance for editing these expressions. This is included below:

```
The regular expression editing window allows one to edit a regular expression
which is used to parse the input file line by line.  These are in a pre-expanded
form which allows usto call certain predefined helper matching functions.  The
'View Expanded' button allows one to view the resulting expanded regular expression.

The predefined matching helper functions are:

    matchInt(name)
    matchFloat(name)
    matchStr(name)
    matchLatLon(name)


  matchInt matches an integer value with an options sign prefix.

  matchFloat matches a float value with an optional sign, optional
  pre-decimal value (eg: .012) and optional post-decimal value (eg: 12.).

  matchStr is used to match a string.  Its only use is to parse the 'partial
  seconds' field seen in the odetics shotlog file.

  matchLatLon matches a lat/long coordinate value in either of the following
  formats:
     +42.157
     42 22.1 N
  Meaning it matches both decimal degree with optional sign,
  as well as degree minute with minute in decimal form, and an optional compass
  direction character (N, E, S, W).  Thus this helper function should match most
  lat/lon coordinates available.
```

One references these functions like so:

    $helperFunction(arg1, arg2)

One should be able to manage input from various types of input files simply by rearraging the order that these helper functions are expected.

Regular expressions are used to 'match' expressions. For example, to match one or more items of whitespace (spaces, tab, etc) one enters:

    \s+

The '\s' means match whitespace, while the + modifier says 'match the previous argument 1 or more times'. Thus the above statement consumes 1 or more pieces of whitespace.

NOTE: the only optional match name in the shot times is 'partialSec' - if not matched, it is not used.

If one is using lat/long coordinates from the shot time data, these need not be in the regular expression match for the navigation data unless gps-gun offset correction is to be performed.

To skip data which is not to be considered by the program, simply enter regular expressions without helper functions. For example, to skip a real value:

    \d*\.\d*

Consumes 0 or more digits, a decimal place, then 0 or more remaining digits.

Advanced Regular Expression Editing

NOTE: These regular expressions are in 'Verbose' mode, meaning that whitespaces/newlines are insignificant, and comments are allowed. (#comment).

Please reference the following URL's for Python regular expression assistance:
  http://www.python.org/doc/current/lib/module-re.html
  http://py-howto.sourceforge.net/regex/regex.html

For the purpose of matching to specific variables (such as shotNo, lat, lon, etc) we use 'Named Groups' - meaning we match a set of input, and tag it with a symbolic name so we can reference it in the loading stage regardless of position. For example to match 'shotNo' assuming it is one or more digits only, the expression would look like:

    (?P<shotNo>\d+)

now with an optional sign (characters within sqaure brackets are a 'set' - matches any):

    (?P<shotNo>[+-]?\d+)

Note that this is the actual way we match integers with matchInt().

For a more complicated example, lets say you have day, hour, and minute all in a field with no space delimeters (but fixed widths) in these formats (day:236, hour:1, min:2):

    2360102

    0010102

To match this we need to use braces {} - this matches {min,max} of the previous elements - since we want a fixed number of digits for each, the min and max are the same.

Then the corresponding regular expression would look like:

  (?P<day>\d{3,3})
  (?P<hour>\d{2,2})
  (?P<min>\d{2,2})

```
If there is the potential for whitespace in between these components instead of
0's, as is the case with many Fortran output files, as follows:

  (dddhhmm)
   236 1 2
     1 1 2
   23456 1
or even
   123 212


(note that the number of spaces cannot be relied upon)

Then a suitable expression to match the above would be:

  (?P<day>[\d\s]{3,3})
  (?P<hour>[\d\s]{2,2})
  (?P<min>[\d\s]{2,2})

This forces a match of exactly 3 (then 2 then 2) repititions of digits or whitespace -
forcing the 'fixed width' nature of the input data.

For more information, please see the URL's above.
```

Modifying:

Please see README.TXT for URL's for modification information – it is also included below:

```
REQUIREMENTS:
-------------

  To run:
  -------
    Python 2.2 or greater,
    wxPython 2.3 or greater
    wxWindows library (included with wxPython in Windows)

    Windows: 95/98/NT/2000/XP (for self-installing .exe): nothing
      To run from source: same dependencies as for Unix.

    Unix machine with Python 2.2, wxWindows and wxPython for running
      from source (see below for windows).  RPM's exist for RedHat
      Linux, and Debian has all needed packages built.


  To Build:
  ---------
    Python 2.2   (www.python.org)
    wxWindows    (www.wxwindows.org)
    wxPython     (www.wxpython.org)
    py2exe       (http://starship.python.net/crew/theller/py2exe/)
    NSIS (For creation of installation .exe -
         http://sourceforge.net/projects/nsis

RUNNING IN WINDOWS:
-------------------
```

```
  To run shotGui (options file editor for backend process):
    One can either run it from the link on the desktop, or from
    the installed directory as shotGui.exe.

    usage: "shotGui.exe [optionalConfigFile.cfg]"

  To run shotCreate (backend shottab creation process):
    Since this is a command line application, this should be run
    from a command prompt to see the output, or from the gui (run
    button) which also displays the output.

    Usage:  "shotCreate.exe configFile.cfg"


RUNNING IN UNIX:
----------------

  While this depends on how this was installed, it is planned that both of
  these utilities be set up in such a way as to allow one to run either
  shotGui/shotCreate by simply typing: (eg: it is installed in the path)

    shotGui  [options.cfg]
    shotCreate  options.cfg

  Where options.cfg is the name of the options file you wish to work with.


BULDING Windows Self-Installing EXE:
------------------------------------

  This requires all the build prerequesites listed above for building in
  windows.

  First, ensure "python.exe" is in your path (eg c:\python22)

  Now, build the shotCreate and shotGui exe's:
    cd to where the .py's are, and type:

      python setup.py py2exe

    This should invoke a process which creates a "dist" subdirectory,
    under which are subdirectories "shotGui" and "shotCreate" - within
    these directories are the corresponding .exe's and some dll's (and
    pyd's).

  The exe's are now built - now to build the self-installing .exe:
    (ensure you have NSIS at least version 2 - prerelease, alpha 7)
    Run "makensis" on the nsi file, such as:

      c:\progra~1\makensis pyShottab.nsi

    This should create a file pyShottab.exe in the current directory -
    this is the self-installing exe that should be distributed for
    all versions of windows.  Note that no other dependencies are
    required for running the program other than this one file.
```

Module Layout:

shotCreate.py – The main conversion program

shotGui.py – The main GUI window

guiComponents.py – components used by shotGui

heading.py – functions for heading interpolation

options.py – module to support reading/writing options to a .ini style file

fileio.py – support for reading extremely large data files, and converting them via a regular expression.

Itpl15.py – simply a string interpolation module that changes "$foo()" into the output from function foo. Written by a 3rd party.

Simple Changes:

Most of the interesting logic is in shotCreate.py and heading.py – these are programmed in a simple fashion, and should be easy to understand and modify.

Should one need to add an option, they would take the following approach:

To add options to be edited there are 3 main components:

1) Add the option to the Options class of options.py – edit the "comments" variable to comment it, and in __init__ add a line such as:
   self.newIntegerVar = 0
   The default type (integer/string/float) must be what you want to use in the program.
2) Modify the GUI options editor so that one is able to edit it's value. Add the gui controls in __init__, and edit menuSave and updateGui methods so that they get loaded/saved properly. A simple example to clone would be total clock drift.
3) Edit shotCreate.py to use the variable, such as "foo = options.newIntegerVar".

Since python is interpreted, one only needs to edit the .py file and run it again to test the new changes – no build step is required. When the new code is well tested, then one can finalize with a self-installing windows .exe for ease of distribution.

Dave LeBlanc
November 29, 2002
(updated September 23, 2004)
Dalhousie University
Dept. of Oceanography