User Interface For Off-line Applications

Sanjay Bhatnagar,
N.C.R.A., Pune

March, 1997

12/2/00

1 Introduction

This document describes the user interface for the GMRT off-line data analysis package. The motivation for developing this user interface system was to have a simple, uniform interface for all the programs that we, as a group, might develop.

Applications using this system can be started in an interactive (the default) or non-interactive mode. Application programs present a list of keywords, the values of which can be set, reset, loaded from or saved to a file in the interactive mode. In the non-interactive mode, these keywords can be set as a list of command line arguments of the form `<KeyWord>=[<Val0>,<Val1>,...]`. While searching for keywords, the interface uses minimum-match algorithm requiring the user to specify only the least number of characters which uniquely identify the keyword of interest. While setting the values of the keywords, the interface also uses `bash` styled command line editing and file name completion mechanisms. The command line editing is normally done using `emacs` commands, but can be configured ('degraded'!) to use `vi` styled commands. The interactive shell also defines some package wide commands (common to all the applications in the package) to provide basic as well as detailed online help.

2 The User interface commands

In an interactive session, the application programs present a list of keywords to which the application is sensitive. The most basic commands in the shell are for setting or resetting the values of the keywords. Keywords can be assigned a list of values, separated by a comma (','). However, not all keywords will require a list of values (though it may not be an error to provide a list of values where a lesser number of values are required). The number of values that are required by a keyword can be found using the command `'?'`. The value of a keyword can be set by a command of type `<KeyWord>=[<Val0>[,<Val1>[...]]]` and can be reset by omitting the values in such commands (command `<KeyWord>=`).

The `help` command provides the basic help about the shell itself and prints following on the screen:

```
Commands in the interactive mode:

Use <Key>=<Val1,Val2,..> to set value(s) for a keyword
Use <Key>=<RETURN> to unset value(s) for a keyword

 inp          : To see the various keywords and their values
 go           : To run the application
 gob          : To run the application in background
 cd           : Change working directory
 help         : This help
 ?            : Information on the type of the keyword
 explain      : Detailed help, optionally of keywords/task [[Key][:Task]]
 save         : Save the values, optionally in a file
 load         : Load the values, optinally from a file
 edit         : Use an editor to (un)set the values
 quit         : Quit the application

Any other input will be passed to the system shell
```

These commands form the standard set of shell commands available from all applications. There may be a few extra commands which may be application dependent and will change from application to application. However, the `help` command will always show all the commands available for a given application.

The command `inp` is for viewing the current settings of the keywords. The `go` command is used to end the interactive session. After this command, the application determines the user defined values of the various keywords and starts execution. The `gob` command is similar to the `go` command except that it will run the application in background and print the Process ID (PID) number of the background process on the screen (also see section 3). The `cd` command is used to change the current working directory within the interactive shell. The command `'?'` provides some basic information about the type and number of values the keywords expect. `explain` command provides as detailed a help about the application and the keywords as the author of the application has cared to write in the help file. These help files are located in the directory specified by the environment variable `GDOC` (see section 3). The command `save` is used to save the current setting of the keywords in a file. By default, these values are saved in a file named `./<Application Name>.def`. To save the values in some other file, the file name can be provided as an argument to this command. The `load` command is the conjugate of `save` - by default it loads the settings for the keywords from the file `./<Application Name>.def`. If this file

is already present when the application is started, it is loaded automatically. Alternatively, it can load settings from a file provided as an argument to the `load` command. The command `edit` lets the user edit the keyword values in an editor of choice specified by the environment variable `EDITOR` (see section 3). The `quit` command is used to quit the interactive session without executing the application.

All inputs to the interactive shell, which are neither any of the above mentioned commands nor any of the application specific commands, are passed to the underlying Operating System (OS) user shell. Hence, most of the native OS commands shall still be available from this shell. (Users however must be aware that certain OS shell commands like "setenv" (for `csh` users) or "export" (for `bash/sh` users) will seem to work, but will not have the desired effect).

As mentioned before, all keywords can take a list of values, each of which is separated by a comma (','). The number of comma separated values that a keyword expects can be found using the `'?'` command which reports the type and the number of the values expected. The number of such values expected will be reported along with the type, enclosed in '[' and ']' pairs. Some keywords might be able to take mixed type of values (e.g., string, floats, integers). For these keywords, the reported type will be "UNKNOWN". For the keywords, which can accept any number of values, the type string will be followed by the string "`[]`".

As mentioned above, comma is treated as a separator in a list of values for a keyword. To suppress its interpretation as a separator, it has to be "escape" using the backslash ('\') before the comma. For example, if the value of keyword `key` has to be set to the a string `''Funny, value with a comma''`, it can be done using

```
key=Funny\, value with a comma
```

Similarly, if any other characters reserved for the command syntax ('\', '[', ']', '=') are to be used as part of the value, they also will be required to be escaped.

Keywords that take numeric values can also take arbitrary mathematical expressions (or a list of them). Any of the following functions or constants can be used in these expressions:

- Functions

  $sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh,$
  $exp, ln, log, log10, sqrt, fabs, floor, ceil, rint$

- Constants

  1. `PI`: value of π
  2. `C`: speed of light in m/s.
  3. `R2D,D2R`: multiplicative constants for conversion from Radians to Deg. and vice versa

4. `H2R,R2H`: multiplicative constants for conversion from Hours to Radians and vice versa
5. `SOL2SID,SID2SOL`: multiplicative constants for conversion from Solar to Sidreal time and vice versa

Numbers in the expression can be in any of the following number representations:

- Integer format, real (float) format
- 1.0E-1 or 1.0e-1 (=0.1)
- 1.0D-1 or 1.0d-1 (=0.1)
- 1h10m0.1s: the time format - converts the number to seconds before using it.
- 1d1'1": The angular format - converts to arc seconds before using it.
  (1d1' = 3660.0"and -1d1' = -3660.0")

2.1 De-referencing mechanism

Keywords are treated as variable by the shell. The user interface system allows the application programmers to have more symbols which are treated as constants (i.e. their values cannot be modified by the user). We refer to these symbols as ``const symbols''. Some applications may load a number of frequently used values of various keywords as constants (for example, the list of symbols reported by the `showfmt` command of the `xtract`[1] program). By default, none of the `const symbols` are presented to the user. Those applications which carry these extra shell constants would also provide an extra shell command to view these variables.

Values can be transfered from one keyword (or a `const symbol`) to another keyword by referring to the source keyword by appending its name to the '$' operator. If the value of the source keyword is a list, the entire list is transfered to the target keyword. If a particular element of the list is to be transfered, it can be indexed by appending the index of the desired value bracketed by '[' and ']'. Particular element of a list of values can also be altered by referring it in a similar manner.

For e.g. to transfer the i^{th} value from keyword `Key1` to a keyword `Key2`, one could use `Key2 = $Key1[i]`

2.2 File name convention

Most off-line applications can perform I/O using UNIX pipes. If the name of the input source begins with the '<' character, the rest of the file name is treated as a command, the output of which becomes the input of the application. For example,

[1] http://langur.ncra.tifr.res.in/~sanjay/Offline/xtract

to supply the output of a program `tmac` as the input of the program `xtract`, the ``in'' keyword of `xtract` must be set to "<tmac". On giving the go command to `xtract`, the interface of `tmac` will be started and the keywords of `tmac` can then be set in the normal fashion. The go command to `tmac` will start the execution of `tmac` who's output becomes the input of `xtract`. Till this time, execution of `xtract` would remain blocked, waiting for input from `tmac`.

Similarly, if the output file name begins with the '|' character, the rest of the file name is treated as a command, the input of which becomes the output of the off-line program.

By convention, a blank output file name implies that the output would go to the standard output stream (usually your screen) and a blank input file name implies that the input would be read from the standard input stream (usually your keyboard).

3 Customization

The user interface can be customized using the following environment variables.

- `GDOC`

  `GDOC` must point to the standard directory where documents explaining the various off-line data analysis programs are kept. The `explain` command will first look for the explanation file in the local directory and then in the directory specified by this variable. The name of the explanation file can be constructed by appending the suffix ``.doc'' to the application name.

- `GERR, GOUT`

  `GERR` and `GOUT` variables are used when the application is run using the `gob` command (see section 2). The standard output of the application will be redirected to the file specified by `GOUT` while the standard error stream will be redirected to the file specified by `GERR`. By default, these variables are set to `/dev/null`.

- `GCONF`

  Some applications may load frequently used setting for some keywords. These values are loaded as constants in the user shell. Their values can be transfered to the application keywords by referring to their value by '$' mechanism (see section 2.1). The path for the file containing these values is specified by the environment variable `GCONF`. The configuration file name is the name of the application with a "`.config`" suffix. If `GCONF` is not defined, the application looks in the directory specified by `GDOC`. If this variable is also not defined, or the configuration file is not found, the application will look for the configuration file in the current directory.

- `EDITOR`

  This environment variable is used to specify the name of the text editor to be used in the `edit` command (see section 2). The default editor is `emacs`.

- `GDEFAULTS`

  The default values of keywords can be saved in a system wide file. Such a file can be automatically loaded by the applications upon startup. The defaults file name is constructed by appending "`.def`" to the name of the application. `GDEFAULTS` variable specifies the directory where this file is to be found.

  By default, the application will look for the defaults file in the current directory.

  If a keyword appears in the "`.def`" as well as the "`.config`" file, the keyword will be treated as a shell constant. This can be used to effectively produce specialized versions of an application program by writing an appropriated "`.config`", where keywords can have fixed values, not alterable by the user (for e.g., a version of `xtract` which will read input from the shared memory of the GMRT data acquisition system).

The following two variables are effective only for versions of the user interface libraries which use the GNU Readline and the History libraries.

- `GHIST`

  `GHIST` specifies the file in which the history of the commands issued in the interactive session are saved. This file will be common to all applications. The default history file is `$HOME/.g_hist`.

  This is also the file from which all applications will load the command history.

- `MAXGHIST`

  `MAXGHIST` should be set to the maximum number of command history entries which the user wishes to save. By default this is set to 100.

3.1 The `help` keyword

The `help` keyword is special (not to be confused with the shell command `help` described in section 2). It is never displayed in the list of keywords, but all applications are sensitive to it. To use it, it must be specified as command-line argument. Following is a list of versions of command-line arguments involving the `help` keyword and their corresponding effects:

- `help=noprompt`

  The application runs in the non-interactive mode. This is useful when the application is run from within a shell script.

When run in this mode, value of all the keywords which needs to be set must be supplied on the command-line (the order of the command-line options is not important). The keywords must be fully spelled in the command-line options (i.e., no minimum-match will be applicable).

- `help=explain`

  This executes the `explain` command of the interactive shell (see section 2) without starting the interactive shell.

  If a keyword is supplied within brackets (`'('` and `')'`) immediately after the `explain` string (for e.g. like `help=explain(out)`, help will be provided for the specified keyword alone. If a application name is also included within the brackets, separated from the keyword by a colon (':'), help for the named keyword of the named application will be provided. If the keyword is skipped (but not the colon), entire help of the named application will be provided.

- `help=doc`

  This results into an empty documentation file written on the standard output, in the required format, with a list of keywords to which the application is sensitive. This is for use by authors of the applications and to encourage them to not only write the documentation, but also in a uniform format.

- `help=dbg`

  Authors of the applications have the facility to have hidden keywords which are normally not displayed for the user, but will be used by the application internally. These keywords can be accessed as normal keywords by setting `help` keyword to the value "`dbg`".

Appendix: Antenna/Baseline naming convention

There is a uniform antenna and baseline naming convention used by most (all?) off-line applications.

A fully qualified antenna name consists of three fields separated by '-' character. The first field is a 3 letter antenna name (for e.g. "C01"). The second field is a 3 letter name for the side band ("USB" for Upper Side Band, and "LSB" for Lower Side Band). The last field is a 3 letter name for the IF ("175" for the 175 MHz IF and "130" for the 130 MHz IF). For e.g.a fully qualified specification for the 130 MHz IF, upper sideband data for C10 would be `C10-USB-130`.

Each of the field can also be replaced with a regular expression. For e.g. a compact way to specify 175 Mhz IF, lower sideband for all central square antennas is `C.+-LSB-130`. Here the '.' in the antenna name matches any character and the '+' operator in the antenna name operates on the '.', one-or-more number of times. Hence, effectively ".+" is equivalent to the wild-card '*' operator.

From this point onwards, "antenna name" implies a fully qualified antenna identification.

A fully qualified baseline names consist of two antenna names separated by colon (':'). The antenna names themselves can be replaced by regular expressions. For e.g. to specify all baselines of `C00-USB-175` antenna with respect to all other central square antennas alone, one could use `C00-USB-175:C.+`. To specify all baselines with arm antennas alone, one could use `C00-USB-175:[EWS].+`.

To select the self correlations, one must use the prefix character 'A' for the antenna names. In such a case, the name of the second antenna is redundant and therefore not required.

Read the manual on the POSIX regular expression syntax and use your creativity to avoid otherwise large amounts of typing (with a proportionately large number of typographic errors and the resulting frustration!).

Examples:

- All baselines with C11

  `baselines=C11`

- Self correlation of C11

  `baselines=AC11`

- Self correlation of C11 130 MHz polarization channel, any side band

  `baselines=AC11-.+-130`

- All USB baselines with C11

  `baselines=C11-USB-.+:.`

or

```
baselines=C11-USB-.+
```

- All USB baselines with C11 and baselines 10,15 and 18

```
baselines=C11-USB-.+,10,15,18
```