The User Interface

S.Bhatnagar

Jan. 23, 1997

[Postscript version][1]

1 Introduction

This document describes the user interface of the offline data analysis package. The motivation behind developing this user interface system is to have a simple, uniform interface for all the programs that we will write.

All applications using this system, when started, will start an interactive shell which will present the list of keywords to which the application is sensitive. The user can then set/reset the values of these keywords before starting the application. The initial values of any of these keywords can also be supplied as command-line arguments in the format <KeyWord>=[<Val0>[,<Val1>[...]]], or loaded from a file by a single command.

Numeric values of any of the keywords can be any algebraic expression.

2 The User Interface Commands

Upon startup, applications will present a list of parameters to which it is sensitive. This will be in the form of various keywords. The values of all these keywords will be empty, unless the keywords was set on the command-line in the format <KeyWord>=[<Val0>[,<Val1>[...]]].

Following are the commands available from this interactive shell:

```
cd       : Change working directory
inp      : To see the various keywords and their values
go       : To run the application
gob      : To run the application in background
help     : This help
?        : Information on the type of the keywords
edit     : Use an editor to (un)set values
save <F> : Save the values in the file <F>
```

[1] http://www.aoc.nrao.edu/ sbhatnag/GMRT_Offline/ui.ps

```
load <F> : Load the values from the file <F>
explain  : Help on the meaning of the various keywords
quit     : Quit the application
```

These commands form the standard set of shell commands that all applications will normally understand. There may be a few extra commands which may be application dependant and will change from application to application. However, the "help" command is always expected to show all the commands that the application can understand.

The keywords can be set/reset from this shell by commands of the type `<KeyWord>=<Val>`. For recognizing the names of the keywords in such commands, the shell uses the minimum match algorithm - minimum

The user interface of all application programs by default presents a list of keywords to which the application is sensitive. This is presented as a list of variables of an embedded interactive shell. The user can set, reset, load, save, etc. the values of these keywords using the basic shell commands in an interactive shell. In general, keywords can be assigned a list of comma-separated values. The number of values that a keyword expects can be found via the shell-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 shell-command `help` 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, optionally 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. However, the user interface libraries provide mechanisms for application programmers to add application specific shell-commands as well as alter the behavior of the basic shell-commands. Hence, there may be a few extra commands which may be application dependent and will change from application to application. However, the help command will automatically always show all the commands available for a given application.

Following is a short description of the shell commands:

- Command inp displays the current setting of the keywords.
- Command go signifies the end of the interactive session. After this commands is issued, the values of the keywords set by the users are used by the application internally and the execution of the main program proceeds.
- Command gob is similar to go except that it will run the application in the background and print the process ID (PID) number of the background process on the screen (also see section 3).
- Command cd changes the current working directory within the interactive shell.
- Command '?' gives some basic information about the type and the number of values the keywords expects.
- Command explain gives 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).
- Command save saves the current setting of the keywords in a file. By default, these values are saved in a file named ./<Application Name>.def. If a file name is supplied as an argument of this command, the values will be saved in the file. this command.
- Command load 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.
- Command edit allows user to edit the keyword values in an editor of choice specified by the environment variable EDITOR (see section 3).
- Command quit quits the interactive session without executing the application.

All inputs to the interactive shell, which are not any of the above mentioned commands or 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 earlier, the values of the keywords can be a list of comma-separated values. The expected number of values will be reported by the shell-command '?' along with the type of the values, enclosed in '[' and '] ' pairs. A blank set of square brackets ("[]") will be printed for keywords which can accept any number of values. Some keywords may accept values of mixed types (e.g., string, floats, integers). For such keywords, the reported type will be "UNKNOWN".

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

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

The characters '\', '[', ']', '=', which are part of the shell-command syntax, must also be escaped similarly to be used as part of the values of the keywords.

Numeric values of keywords can be arbitrary mathematical expressions. Following functions and constants can be used in these expression:

- Functions

  sin,cos,tan,asin,acos,atan,atan2,sinh,cosh,tanh,
  exp,ln,log,flog10,sqrt,fabs,floor,ceil,rint

- Constants

  1. `PI`: value of π
  2. `C`: speed of light in meters per second.
  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 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

The user interface system allows the application programmers to define shell symbols which are treated as constants (i.e. their values cannot be modified by the user). These symbols are referred to as the `const symbols`. Some applications may load a number of frequently used values of various keywords as `const symbols` (e.g., the list of symbols reported by the `showfmt` command of the `xtract`[2] program). However, by default, none of the `const symbols` are presented to the user and its the responsibility of application program to add extra shell-commands for viewing the list of `const symbols`.

Values can be transfered from one keyword or a `const symbol` to another keyword. E.g. to transfer the i^{th} value from keyword `Key1` to a keyword `Key2`, one could use `Key2 = $Key1[i]`. `Key2=$Key1` will transfer all values of `Key1` to `Key2`.

[2] http://www.aoc.nrao.edu/ sbhatnag/GMRT_Offline/offline.html

2.2 File name convention

Most off-line applications can perform I/O using UNIX pipes. If the input file name begins with '<', the rest of the file name is treated as a command, the output of which becomes the input of the application. For example, 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`". After the shell-command `go` is issued for `xtract`, the interface of `tmac` will be started and the keywords of `tmac` can then be set in the normal fashion. When the shell-command `go` is issued again, `tmac` will begin execution, and the output of `tmac` 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 is 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 of the interactive shell and can be stored in a configuration file. Their values can be transfered to the application keywords by referring to their value by '$' mechanism (see section 2.1). The path of the directory containing these configuration files is specified by the environment variable `GCONF`. The configuration filename is constructed by appending the suffix "`.config`" to the application name. 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`

  `EDITOR` 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 "defaults file". `GDEFAULTS` variable specifies the directory where these files are stored. Such a file can be automatically loaded by the applications upon startup. The defaults file name is constructed by appending the suffix "`.def`" to the application name.

  By default, the application will look for the defaults file in the current directory. If a keyword appears in the "`.def`" as well as in 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 (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 is 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

All applications are sensitive to the keyword `help` (not to be confused with the shell command `help` described in section 2). It is never displayed in the list of keywords. To use it, it must be specified as a command-line argument. Following is the list of values that can be set for this keyword on the command-line and their effect:

- `help=noprompt`

  The application runs in the non-interactive mode. This is useful when the application is run from 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 (e.g. `help=explain(out)`, help will be provided for the specified keyword alone. If an application name is also included within the brackets, separated from the keyword by a colon (':') (e.g. `help=explain(out:xtract)`, 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`

  Application programmers can define 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`".