

The User Interface

Introduction

All communication with the user in XSPEC is performed through the tcl user interface. When XSPEC starts, a tcl interpreter is initialized, and the XSPEC commands are added to it so that the tcl interpreter understands them. The XSPEC commands, which are C++ functions, define the syntax through a new built-in library of utility functions. The parser used in earlier versions of XSPEC has been discontinued: however the syntax understood by XSPEC12 is much the same as before.

XSPEC and tcl/tk

Because tcl is a full scripting language, users can write complex scripts with loops, branching, etc., which utilize XSPEC commands. Here we describe how to use those features of tcl necessary to give the user similar functionality to that available in previous versions of XSPEC, and to give information on the details of our tcl implementation that may be useful to experienced tcl users. For a description of tcl, see, for example, *Practical Programming in Tcl and Tk*, B. Welch, (1997, Prentice Hall).

Tk, tcl's companion graphical user interface (GUI) toolkit, is also loaded by XSPEC on startup. It is planned that future versions of XSPEC will provide an optional GUI side-by-side with the command line interface (CLI). Although XSPEC does not currently use tk, its presence allows users to write XSPEC scripts with graphical interfaces using Tk commands.

A note on command processing

To emulate the performance of the former XSPEC parser, the command functions are programmed to react similarly to some of its features.

The # sign is used for comments in tcl, but may appear only at the beginning of a command. tcl and XSPEC both ignore carriage returns on a new line, but XSPEC also ignores the skip character ' / '. The character sequence ' /* ' entered during a command exits that command, sets any responses to the default response, and returns the user to the prompt. The ' \ ' character is used in tcl for continuing a command onto the next line.

Additionally, note that in tcl, commands and their arguments are delimited by white space. They are terminated by a newline or semicolon, *unless* there is an open set of parentheses ' { } ' constituting a loop or test structure. In other words, in tcl the following starts a loop:

```
while { condition } {
...
}
```

but

```
while { condition }
{
}
}
```

is incorrect, since in the latter case the while command terminates at the end of the first line.

Command Recall/Editing

The XSPEC/tcl interface also uses gnu readline for command input, which allows command line editing and interactive command recall. On most systems, the left and right arrow keys and the backspace/delete key can be used to navigate and edit the command line. The up and down arrow keys can be used to step thru the command history list. Gnu readline is highly customizable, and many more editing/recall functions are available. Readline documentation can be generated in either postscript or html format from the files in the xanadu/readline/doc directory distributed with the source.

The default implementation of tcl also supports a C-shell like command recall mechanism. The `history` command gives a numbered list of the most recently entered commands. Any command in the list can be re-executed by entering `!n`, where `n` is the number of the command in the history list. The previous command can be re-executed by entering `!!`. The most recent command that begins with a string can be re-executed by entering `!prefix`, where `prefix` is the string the command begins with.

Note that command recall is implemented using the tcl `unknown` procedure, part of which is a script file loaded by tcl at run time. See the section on the `unknown` command for more details on how it is implemented in XSPEC.

Logging

The `log` command can be used to open a log file to which all input and output to tcl will be written. Reading these log files can potentially be confusing when logging tcl flow control commands such as `while` or `for`. This is because tcl treats the body of these commands as an argument of the command. Thus when the command is echoed to the log file, the entire body of the command is echoed with it.

In order to make this situation less confusing, before commands are echoed to the command file, all newline characters are replaced by semicolons, and the resulting command line is truncated to 80 characters. Then any commands executed within the body of a flow control command are echoed as they are executed.

Consider the following sequence of tcl commands within XSPEC:

```
XSPEC12> log
Logging to file: xspec.log
XSPEC12> set i 1 ; set product 1
1
XSPEC12> while {$i <= 5} {
XSPEC12> set product [expr $product * $i]
XSPEC12> incr i
XSPEC12> }
XSPEC12> set product
120
XSPEC12>
```

This would produce the following output in the file `xspec.log`:

```
Logging to file: xspec.log
XSPEC12> set i 1
set product 1
```

```

1
XSPEC12> while {$i <= 5} {;set product [expr $product * $i];incr i;}
    expr $product * $i
    set product [expr $product * $i]
    incr i
    expr $product * $i
    set product [expr $product * $i]
    incr i
    expr $product * $i
    set product [expr $product * $i]
    incr i
    expr $product * $i
    set product [expr $product * $i]
    incr i
    expr $product * $i
    set product [expr $product * $i]
    incr i
XSPEC12> set product
120
XSPEC12>

```

Command Completion

tcl attempts to match the name of any entered command as an abbreviation of a valid command (either a tcl or XSPEC command). If the entered command matches more than one valid command, tcl then lists the possible choices, but does not execute the command. For XSPEC commands, aliases have been constructed matching the command to its minimum abbreviation, as listed when typing `?' at the XSPEC prompt (see under Aliases).

For example, the minimum abbreviation for the `plot' command is `pl'. Thus, typing `pl' will execute the plot command, even though this would otherwise be ambiguous with the tk command `place'. Command completion is also implemented using the tcl `unknown` procedure, part of which is a script file loaded by tcl at run time, and may be different or not exist on your system. See the section in this help file on the `unknown` command for more details on how it is implemented in XSPEC.

N.B. tcl explicitly switches off command completion for scripts. Because of the way scripts are implemented in XSPEC, however, command abbreviations nevertheless do work in scripts entered with the `@` command, but not when entered from the command line or using the source command. See below for more details about tcl scripting.

Unknown Procedure

tcl provides a facility whereby if it cannot match an entered command to its list of known commands, it calls the `unknown` procedure, with the unmatched command (along with its arguments) as its argument. The version of `init.tcl` distributed with tcl contains a version of the `unknown` procedure. When tcl initializes, it looks in several standard places for a script file named `init.tcl`, which it executes if found. The `unknown` procedure is where tcl does command completion and automatic shell command execution.

At start up time, XSPEC loads its own `unknown` procedure, , which it uses to intercept script processing requests of the form

```
XSPEC12>@<script>
```

and renames the previously defined `unknown` procedure to `tclunknown`. If XSPEC is not doing any special processing, it simply passes any unmatched commands on to `tclunknown`, which then processes them as usual. XSPEC has its own special version of the `unknown` procedure

These factors need to be taken into consideration for programmers writing tcl scripts for use within XSPEC. For example, if after initialization, users wishing to load a different version of the standard tcl `unknown` procedure should name that procedure `tclunknown`, rather than `unknown`.

Aliases

Command name aliases can be constructed using the tcl `interp` command:

```
interp alias {} <command_alias> {} <xspec_command>
```

where `<xspec_command>` is the name of the command you wish to make an alias for, and `<command_alias>` is the name of the alias you wish to set for the command. The `{ }` are *required* syntax.

To delete the alias `<command_alias>`, simply nullify it with:

```
interp alias {} <command_alias> {}
```

Initialization Script

When running interactively, the user has the option of providing an initialization script, which will be executed after XSPEC completes its startup procedure, ie. just before it begins prompting for commands. The file should be named `xspec.rc` and located in the directory `$HOME/.xspec`. **If one runs XSPEC in batch mode, by specifying a script on the command line, this initialization script is not executed.**

When multiple users are accessing a single system-wide XSPEC build, the installer can also provide additional initialization that will apply to all users. See the section “XSPEC Overview and Helpful Hints: Customizing XSPEC” for more details.

XSPEC Command Result

After being executed, many tcl commands return a result string, which is echoed to the terminal when the command is entered on the command line. When writing complex tcl scripts, this result can be stored and/or used as a test in loops, etc. When XSPEC commands are executed, they write information to the terminal by writing directly to the appropriate output channel. However, when running interactively, the tcl result string is also written to the terminal after the command is executed. The `tclout` command (see command description) creates tcl variables from `xspec`’s calculations.

Script Files

XSPEC/tcl script files can be executed in three different ways, as follows:

<code>xspec - <script></code>	executing script on initialization
<code>XSPEC12>@ <script></code>	executing script from within the program
<code>XSPEC12>source tclscript</code>	use tcl's source command from within the program.

Each of these usages does something slightly different. In the first form, XSPEC will execute a file called `<script>`. One may execute a series of script files at startup with the following command syntax:

```
unix> xspec - file1 file2 file3 ...
```

Note that the space following the `-` is required.

The second form is `@<name>`, where `<name>` is the name of the script file to be executed. Here the default extension of `.xcm` is assumed. Scripts containing valid tcl or XSPEC commands will be executed using this form, and (unless the script ends in **quit** or **exit**) will return to the interactive prompt after completion.

The final form, using tcl's source command, is intended for the special case where the script contains the implementation of a new command written in tcl/tk. See the section on **writing custom commands** for more details. In current tcl versions it compiles the script into bytecode representation for more efficient execution, and adds any procedures defined in the script to the set of commands understood by the interpreter. **It will not work for general scripts containing XSPEC/tcl commands, for example those produced by XSPEC's save command. These should rather be executed using the @ form.**

Note that only in the second case `@` is there a default filename suffix (i.e. `.xcm`) for both the other methods of script execution the filename must be given in full.

tcl internally switches off the mechanism that expands command abbreviations when scripts are executed. If this were not done, the user could specify command abbreviations that change the behavior of the tcl command set (e.g. `set` for the **setplot** command would redefine tcl's command for setting variables). This behavior can be overridden with the statement

```
set tcl_interactive 1
```

near the beginning of the script, but it is not recommended to do so. Instead, we strongly recommend spelling out command names in full within XSPEC scripts.

Command Echoing

By default, when XSPEC is executing a script file, it echoes each command to the terminal before it is executed. This can be controlled using the tcl variable `xs_echo_script`, whose default value is 1. If this variable is set to 0, the commands from the script file will not be echoed to the terminal.

Summary

In summary, we suggest the following convention:

- Running an `xspec` script from the unix command prompt is intended to be used for background processing or overnight batch jobs. Using the unix `at` command, one can arrange to receive the log file by e-mail.

- The `@` usage is intended for processing previously run `xspec` command sequences, such as are produced by the **save** command.
- The `source` usage, as well as executing the commands in the script, performs the equivalent of pre-compiling the script for later invocation. Its most appropriate use is in preparing new custom XSPEC command procedures. Once the script is working correctly, it can be placed in the user script directory and become part of the user's standard command set. For examples, see the scripts `addline.tcl` and `modid.tcl` in the directory

`$SPECTRAL/scripts`

that implement the commands **addline** and **modid**. These also show how to make commands self-documenting.

Unix Shell Commands

Shell commands can be executed within XSPEC using the `exec` command (see the help entry on the `exec` command). When running interactively, if `tcl` cannot find a command that matches that entered on the command line, it will search for a shell command that matches the entered command. If it finds a match, it automatically executes the shell command via `exec`. Note that this feature is implemented using the `tcl unknown` procedure, part of which is a script file loaded by `tcl` at run time, and may be different or not exist on your system. See the section in this help file on the `unknown` command for more details on how it is implemented in XSPEC.

Note that the `tcl exec` command executes the given command directly, without first passing it on to the shell. Thus no globbing (ie. expansion of wildcards such as `*.pha`) is performed. If you wish to pass your command through a shell for wildcard expansion, etc, use the `syscall` command.

If you want to start a subshell from within XSPEC, simply type the command for starting that shell, ie. type

```
XSPEC12>csh
```

in order to start a C-shell. Note that typing

```
XSPEC12>exec csh
```

will not work properly.

Giving the

```
XSPEC12>syscall
```

command with no arguments will start a subshell using your current shell (`csh`, `tcsh`, `bash`, `sh`, etc).

Writing Custom XSPEC commands

XSPEC commands can be written by users as `tcl` procedures, which have similarities with `fortran` subroutines. Within XSPEC, `tcl` procedures can take arguments and execute XSPEC and `tcl` commands. The syntax for specifying arguments to a `tcl` procedure is as follows:

```
proc my_proc {arg1 arg2}{
...
}
```

```
data 1:1 ${arg1}_s0_20
data 2:2 ${arg2}_s1_20
...
}
```

Here, `arg1`, `arg2` are values supplied by the user (here, part of a filename) from the command line, and substituted wherever `${arg1}`, `${arg2}` appear within the script. One may also give an argument a default value, so that the command so created may be invoked even without needing to specify the argument:

```
proc my_proc {arg1 {arg2 file2} } {
...
}
```

Note that the parentheses enclosing both `${arg2}` and `file2` in this expression distinguish this from the case where 3 arguments are required for `my_proc`. Once this file is created, it needs to be `source'd` once, which compiles the script into an internal bytecode representation (this is similar to the way Java operates). Alternatively, one may place it in the user script directory and create an index in that directory, after which case it will be found automatically and compiled the first time it is invoked.

The user script directory is given by the line

```
USER_SCRIPT_DIRECTORY:
```

in the `Xspec.init` file that is copied into `$HOME/.xspec` when the user starts `xspec12` for the first time (the supplied default value for this directory is the `$HOME/.xspec` directory itself). After the script is placed there, perform the following command

```
%xspec12
XSPEC12>cd <USER_SCRIPT_DIRECTORY>
XSPEC12>auto_mkindex .
XSPEC12>exit
```

This will instruct `XSPEC` to build an index of scripts to be loaded on `xspec` startup.

On the next invocation of `XSPEC`, the script will be sourced on startup and will appear in the list of commands `XSPEC` understands.

The `my_proc` procedure is then defined such that one may type:

```
XSPEC12>my_proc eso103 eso104
```

and the data statement in the above example will be executed as if the following had been entered:

```
data 1:1 eso103_s0_20
data 2:2 eso104_s1_20
```

The `tcl info` command can be used to show which procedures have been defined:

```
XSPEC12>info commands <procedure name>
```

This will return `<procedure name>` if that procedure has been compiled (source'd) already or is a built-in command, or nothing if it has not (yet) been invoked or defined.

Scripting commands that prompt the user

The commands **model**, **editmod**, **addmod**, **newpar**, and **fakeit** may prompt the user for more information when used interactively. In order to write scripts that use these commands, one must know how to force XSPEC to enter the information that would be prompted for. The technique is exemplified as follows. Suppose we defined a procedure `xmodel` that makes a model with certain predefined parameter values:

```
set p1 {1.5 0.001 0 0 1.E05 1.E06}
set p2 {1 0.001 0 0 1.E05 1.E06}
proc xmodel {modelString param1 param2 args} {

    model $modelString & $param1 & param2 & /*

}
```

Here the "&" character is taken by XSPEC as a carriage return, delimiting the model string and parameter arguments into separate input lines.

The procedure `xmodel` may be compiled with the command

```
XSPEC12> source xmodel.tcl
```

This creates `xmodel` as a command with two arguments which sets subsequent parameters to their default values. It can be invoked e.g. by

```
XSPEC12>xmodel {wa(po + peg)} $p1 $p2
```

Note that the model string, which contains spaces, needs to be entered in `{ }` or double quotes. Note also that tcl understands a single string argument, `args`, as in

```
proc tclscript { args } {
...
}
```

to mean a variable number of arguments to a procedure (it is supplied as a tcl list, which can be split within the procedure into separate strings for digestion by xspec if present).

Script Example

In the directory `$HEADAS/.../spectral/session` is a script file called `tclex.xcm`. This script gives an example of how one might use the power of tcl's scripting language in an XSPEC session. This script should be executed with

```
XSPEC12> @tclex
```

```
# This script gives an example of how one might use the power of tcl's
# scripting language in an XSPEC session. In this example, XSPEC loops
# thru 3 data files (file1, file2 and file3) and fits them each to the
# same model `wabs(po+ga)'. After the fit the value of parameter 4 (the
# line energy for the gaussian) for each data set is saved to a file.

# Keep going until fit converges.
```



```

query yes

# Open the file to put the results in.
set fileid [open fit_result.dat w]

for {set i 1} {$i < 4} {incr i} {

# Set up the model.
  model wabs(po+ga) & /*

# Get the file.
  data file$i

# Fit it to the model.
  fit

# Get the values specified for parameter 4.
  tclout param 4
  set par4 [string trim $xspec_tclout]

# Turn it into a Tcl list.
  regsub -all { +} $par4 { } cpar4
  set lpar4 [split $cpar4]

# Print out the result to the file.  Parameter value is
# the 0th element of the list `lpar4'.
  puts $fileid "$i [lindex $lpar4 0]"

}

# Close the file.
close $fileid

```

The user is encouraged to read the voluminous on-line documentation and literature available about tcl in order to benefit fully its flexible command processing, graphical interfacing, and scripting capabilities. See <http://www.tcl.tk> for much more information and extensive bibliography.