APS Tcl/Tk Library and Interpreter Extensions

Claude Saunders, Michael Borland
Operations Analysis Group
Accelerator Systems Division
Argonne National Laboratory
September 18, 1996

1. Introduction

This document serves as a User's Manual and Reference for the library of Tcl and Tk procedures produced by the Operations Analysis Group. This document is not always kept up-to-date with the latest installed library. To peruse the latest set of procedures and their usage, use the "**apsHelp**" application by typing "apsHelp &". This document does, however, provide more thorough explanations and examples, and so is useful for the beginner.

1.1. Tk Library

This library is a collection of widget procedures for creating Tk applications with a consistent look-and-feel. To access this library, simply add to your Tcl auto_path variable as follows:

```
set auto_path [linsert $auto_path 0 /usr/local/oag/apps/lib/$env(HOST_ARCH)]
set auto_path [linsert $auto_path 0 /usr/local/oag/lib_patch/$env(HOST_ARCH)]
```

Tcl/Tk will dynamically load the neccessary procedures when you reference them in your script. A basic, functioning skeleton application is as simple as this:

```
#!/bin/sh
# \
exec oagwish "$0" "$@"

set auto_path [linsert $auto_path 0 /usr/local/oag/apps/lib/$env(HOST_ARCH)]
set auto_path [linsert $auto_path 0 /usr/local/oag/lib_patch/$env(HOST_ARCH)]
APSApplication . -name MyMainApplication -version 1.0 -overview NotMuch \
    -contextHelp "This is a general APS widget demo application"
```

The widget procedures in this library all follow a consistent calling convention. All arguments except the first are optional and non-positional. With this capability, the authors of the library can add arguments (options) to the procedures over time without breaking existing applications.

Every procedure has a help procedure of the same name with the suffix “Help”. These help procedures return a usage string to aid the programmer. Context help can be provided for each widget, or for groups of widgets in an application. At runtime, the user of your application can access this help via the Help menu or the keyboard Help key.

1.2. Tcl Library

This library is a collection of non-graphical Tcl procedures to aid in creating standardized applications. To access this library, the same path as the Tk library is used:

```
set auto_path [linsert $auto_path 0 /usr/local/oag/apps/lib/$env(HOST_ARCH)]
```

1.3. Interpreter Extensions

The standard tcl/tk interpreter used by APS operations is called “oagwish”. This interpreter incorporates the public extensions BLT and Tcl-DP, and the local extension, tclCa. The tclCa extension provides the “pv” command set for accessing process variables from within a tcl script. Soon to be added is the tclSDDS extension for accessing SDDS files.

2. Tk Library

The calling conventions for the Tk library are described first, followed by a detailed description of each procedure. A code example is provided for each procedure which may refer to other procedures. Every procedure in this library begins with upper-case **APS**.

2.1. Calling Conventions

Every procedure is as follows:

```
APSWhatever <widget> [<option-list>]
    where <option-list> is a list of
        -name value
    pairs.
```

The procedure creates a widget named <widget> according to the specifications in the <option-list>. The options can be given in any order. For every procedure above, there is a help procedure:

```
APSWhateverHelp
```

This help procedure returns a string with a usage statement. The usage statement describes the accepted <option-list> and lists the widget(s) that are created by the procedure.

Options common to most procedures are described below:

- -parent <widget>
  Pack widget named in first argument into <widget> using the default packing options. If this option is absent, procedure attempts to make a toplevel widget.
- -noPack 1
  Do not pack widget into parent, just create it.
- -packOption <list>
  When widget is packed into parent, use packing options in <list>.
- -contextHelp <string>
  Display <string> whenever this widget is selected in context help mode.

2.2. Procedures

Procedures in this library are grouped by their general purpose:

- Main Windows
- Basic Widgets
- Complex Widgets

2.2.1. Main Windows

The Main Window widgets are the primary windows which a user of an application would interact with. The APSApplication window is complete with a menubar. For interaction via a dialog, APSDialogBox is provided. For a window which is neither a full application, nor a dialog box, APSWindow is provided. Other utility windows are APSAlertBox, APSInfoWindow, APSExecLog, APSFileSelectDialog, and APSFileDisplayWindow.

APSApplication widget

```
Creates basic application framework, setting various global options such as
widget colors. Creates a menubar with File and Help entries. A “userFrame” is
returned into which the programmer may pack application specific widgets.
Options:
-name <string>          Title on window of application.
-version <string>       Version number put into Help/Version menu entry.
-overview <string>      Overview put into Help/Overview menu entry.
-contextHelp <string>
Creates:
    $widget.
    userFrame
    menu.file.menu
    menu.help.menu

Example:
# Note that . should be used as the first application widget name.
APSApplication . -name MyMainApplication -version 1.0 -overview NotMuch \
  -contextHelp “This is a general APS widget demo application”

APSMenubarAddMenu .edit -parent .menu -text Edit
.menu.edit.menu add command -label “Print this” -command {
    puts “this is a new addition to the menu bar.”
}
# Note that I am having this button packed into the “userFrame” created
# by the call to APSApplication.
APSButton .mybutton -parent .userFrame -text Hello -command “puts Hello”
```

APSStandardSetup

```
Takes no arguments. Configures various global Tcl/Tk attributes and colors. This
is executed for you by the APSApplication procedure, so is rarely needed.
```

APSMenubar widget

```
This procedure is generally not used directly. The APSApplication procedure takes
care of producing a standard menubar for your application.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-name <string>
-version <string>       Version number put info Help/Version menu entry.
-overview <string>      Overview put into Help/Overview menu entry.
-contextHelp <string>
Creates:
    $parent$widget.
    file.menu
    help.menu
```

APSMenubarAddMenu widget

```
Adds a new menu to an existing menu bar. Entries may be added to the returned
widget via the “add command” widget command.
Options:
-parent <widget>
-text <string>
-packOption <string>
```

```
-underline 1
-contextHelp <string>
Creates:
    $parent$widget.
    menu

Example:
APSMenubarAddMenu .edit -parent .menu -text Edit
.menu.edit.menu add command -label "Print this" -command {
    puts "this is a new addition to the menu bar."
}
```

APSDialogBox widget

Creates a standard dialog box framework with OK and Cancel buttons, and a "userFrame" in which to pack dialog specific widgets. The programmer should configure the command for each button according to their needs, although both OK and Cancel button commands default to destroying the dialog window. The OK button is initially disabled, and must be explicitly enabled by the application. The idea here is that the OK button should not be enabled until the user has typed in sufficient information to consider the dialog "completed". If possible, the cancel button command should undo any input the user may have begun.

```
Options:
-parent <widget>
-noPack 1
-packOption <list>
-name <string>
-width <string>
-height <string>
-contextHelp <string>
Creates:
    $parent$widget.
    userFrame
    buttonRow.ok.button
    buttonRow.cancel.button

Example:
APSDialogBox .box1 -name DialogBox \
  -contextHelp "This is the demo dialog box."

# Change the default command for the Cancel button.
.box1.buttonRow.cancel.button configure -command
  {puts "this replaces the previous cancel command"}

# Add a new button.
APSDialogBoxAddButton .new -parent .box1 -text "Enable OK" -command \
  {APSEnableButton .box1.buttonRow.ok.button} -contextHelp \
     "To demonstrate, this button enables the OK button"

# Put a message widget into the "userFrame" portion of the dialog box.
message .box1.userFrame.message -justify left -text [APSDialogBoxHelp] \
      -width 300
pack .box1.userFrame.message -fill both -expand 1
```

APSDialogBoxAddButton widget

Adds a button to the bottom row of buttons created by a call to APSDialogBox.

```
Options:
-parent <widget>
-text <string>
-command <script>
-contextHelp <string>
Creates:
    $parent.buttonRow$widget

Example:
See the example for APSDialogBox.
```

APSExecLog widget

Executes an arbitrary Unix command and displays the output in a scrollable window. A script may be provided with the -callback option. This script is

```
executed upon successful completion of the unix command. If APSExecLog is called
again with the same widget name, the window will be raised and reused.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-name <string>
-width <string>
-height <string>
-unixCommand <string>
-callback <script>
-contextHelp <string>
Creates:
    $parent$widget.
    userFrame
    userFrame.text.text
    buttonRow.ok.button
    buttonRow.cancel.button
    buttonRow.print.button
    buttonRow.enscript.button

Example:
APSExecLog .cmd -unixCommand "sddsplot -col=t,V file.sdds" -callback {
  puts "this is printed upon completion of the command"
}
```

APSInfoWindow widget

```
Creates a window for displaying a general informational message for the user.
Window is modeless by default (ie. user can continue interacting with calling
application). A -modal option is provided to block the calling application.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-name <string>
-infoMessage <string>
-modal 1
-contextHelp <string>
Creates:
    $parent$widget.
    msg
    buttonRow.ok.button

Example:
APSInfoWindow .info -infoMessage "The accelerator is running just dandy."
```

APSWindow widget

```
Creates a standard blank window framework with a Close button, and a "userFrame"
in which to pack your widgets.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-name <string>
-contextHelp <string>
Creates:
    $parent$widget.
    userFrame
    buttonRow.close.button

Example:
APSWindow .appWindow1 -name "Application Window"
# Add a scrolled text widget to userFrame
APSScrolledText .stuff -parent .appWindow1.userFrame
.appWindow1.userFrame.stuff.text insert end "This text is in scrolled window."
```

APSAlertBox widget

```
Creates a modal alert box. Interaction with the main application is suspended
until this alert box is dismissed. Useful for notifying users of serious errors.
```

```
Options:
-parent <widget>
-noPack 1
-packOption <list>
-contextHelp <string>
-errorMessage <string>  Display <string> in alert box.
-modeless 1     Do not suspend interaction with main application during dialog.
Creates
    $parent$widget.
    msg
    buttonRow.ok.button

Example:
# The alert box is a toplevel window, so you don't use -parent option.
APSAlertBox .alert -errorMessage "This is a demo error message. Note that int
eraction with the application is suspended until OK is pressed."
```

APSFileSelectDialog widget

```
Creates a dialog box for browsing the file system and selecting a file. The
procedure returns the selected file (and path), or a null string if the user
cancels the dialog. A starting directory may be provided via the -listDir option.
By default, the current working directory will be used.A "glob-style" filter may
be provided via the -listFilter option.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-listDir <string>
-listFilter <string>
-contextHelp <string>
Returns selected file, or null string if dialog is cancelled.

Example:
APSFileSelectDialog .fileselect -contextHelp \
  "Click to select a file, or to change to another directory."
```

APSFileDisplayWindow widget

```
Creates a simple scrolled text window which displays the contents of the file
given by the -fileName option.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-comment <string> This is placed in the window title bar.
-fileName <string> Desired file to display.
-deleteOnClose 1   The file will be deleted when Close button pressed.
-contextHelp <string>
Creates:
    $parent$widget
    userFrame
    userFrame.file.text
    buttonRow.close.button

Example:
APSFileDisplayWindow .fileWin -comment "My File" -fileName file.txt
```

APSScrolledListWindow widget

```
Creates a window with a scrollable list of user-supplied items. Any combination
of items may be selected. Using <Ctrl-Click>, one may select disjoint items in
the list. When either the Close or Accept buttons are pressed, the user supplied
variable is set to a list of the selected items. The Accept button allows the
selection to be accepted without closing the window.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-height <string>
-name <string> Window name
-label <string> Label placed above scrolled list
-itemList <list> Put these strings on scrolled list
```

```
-selectionVar <string> When Accept or Close is pressed, var is set to selection.
-callback <procedure> When Accept or Close is pressed, <procedure> is invoked
with one argument, a list of selected items.
-contextHelp <string>
Creates:
    $parent$widget.
    userFrame
    userFrame.sl.listbox
    buttonRow.close.button
    buttonRow.accept.button

Example:
# Note: two examples follow, one using -selectionVar, and one using -callback
set mySelection ""
APSScrolledListWindow .slw -name MyList -itemList {one two three four} \
  -selectionVar mySelection -label "Select one or more from list"
# You can wait for selection using tkwait
tkwait variable mySelection

# Alternately, you can use the -callback option
proc myCallback {list} {
  puts "You selected: $list"
}
APSScrolledListWindow .slw2 -itemList {one two three four} -callback myCallback
```

2.2.2. Basic Widgets

These form the foundation of the interior of a Tk application.

APSButton widget

```
Creates a single button. The command option specifies the script to be executed
when the button is pressed. The highlight option creates a button with a black
outline.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-text <string>          Text on button.
-command <script>       Script executed on button press.
-highlight 1            Highlight the button (indicating primary).
-size <string>          where <string> is small or medium (default)
-contextHelp <string>
Creates:
    $parent$widget.
    button

Example:
APSButton .b -parent .userFrame -text "Press Me" -command "puts pressed"
```

APSEnableButton widget

```
When passed button created above, button and its command are enabled.
```

APSDisableButton widget

```
When passed button created above, button and its command are disabled.
```

APSFrame widget

```
Creates an (optionally) titled, visible frame.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-label <string>
-width <string>
-height <string>
-orientation <string>   where <string> is horizontal or vertical (default).
-geometry <string>      accepts X geometry string when a top level window
-contextHelp <string>
```

```
Creates:
    $parent$widget.
    label
    frame

Example:
APSFrame .fr -parent .userFrame -label "Empty Box" -width 80 -height 80
```

APSFrameGrid widget

```
Creates an untitled x by y grid of frames where x is the number of elements in
the -xList option and y is the number of elements in the -yList option. Either
xList or yList may be omitted to create a vertical or horizontal stack of frames.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-xList <list>  list of names to be given to frames in x direction
-yList <list>  list of names to be given to frames in y direction
-width <string> width of each frame in grid
-height <string> height of each frame in grid
-relief <string>  standard Tk reliefs (flat, ridge, raised, sunken, groove)
-bd <string> width of relief border
-contextHelp

If you supply just -xList or just -yList, it creates:
$parent$widget.
        <name1>,...,<namen> where names are from xList or yList

If both -xList and -yList specified, it creates:
$parent$widget.
        <xname1>.<yname1>,...,<xname1>.<ynamen>
        ...
        <xnamen>.<yname1>,...,<xnamen>.<ynamen>
 <string>

Example:
APSFrameGrid .fg -parent .userFrame -xList {a b c} -yList {x y z} \
   -relief ridge
```

APSLabeledEntry widget

```
Creates a standard labeled text entry widget. Text entered in the widget will set
the variable given by -textVariable option.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-label <string>     Label to left of entry widget.
-textVariable <variable>    Variable associated with text entry widget.
-width <string>
-contextHelp <string>
Creates:
    $parent$widget.
    label
    entry

Example:
set filename ""
APSLabeledEntry .filename -parent .userFrame -label File: -width 40 \
  -textVariable filename -contextHelp "Enter file name here"
```

APSLabeledOutput widget

```
Creates a standard labeled text output widget. Text displayed is taken from the
variable given by -textVariable option.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-label <string>             Label to left of entry widget.
-textVariable <variable>    Variable associated with text output widget.
```

```
-width <string>
-contextHelp <string>
Creates:
    $parent$widget.
    label
    entry

Example:
set directory "/usr/bin"
APSLabeledOutput .dir -parent .userFrame -label Dir: -textVariable directory
```

APSUnmap widget

```
Unmaps any widget from the current window. All display information is retained.
Widget can be quickly restored to window with APSRemap. These procedures are
useful for implementing a "More Info..", "Less Info.." dialog.
```

APSRemap widget

```
Remaps any previously unmapped widget.
```

2.2.3. Complex Widgets

These widgets contain multiple widgets and are often composed from the basic widget set.

APSMenuFrame widget

```
Builds a hierarchical menu from the specification given by -menuList option. The
menu provides the ability to bring up applications, execute commands, and to
bring up submenus. Buttons which bring up applications can be restricted to a
given set of users. All buttons can have context help as well. A companion
procedure, APSMenuFrameMake, is provided to ease the building of the "menuList"
specification.
Options:
-parent <string>
-noPack 1
-packOption <list>
-menuList <string>
-parentLabel <string>   label on window
-subMenu 1              not normally used at application level
-contextHelp <string>

Example:
# First define a submenu called "SDDS utilities"
# Note: each call creates one new entry or label of the menu.
APSMenuFrameMake sddsMenu -type label -text "SDDS utilities"
APSMenuFrameMake sddsMenu -type secureExec \
  -text "Quick Monitor" -executable quickMonitor \
  -contextHelp "Brings up the quickMonitor screen, which permits doing simple
EPICS data logging." \
  -allowedUsers "borland saunders"

# Then define the main menu.
APSMenuFrameMake menuList -type label -text "OAGapps"
APSMenuFrameMake menuList -type menu -text \
  "SDDS Utilities" -menuList $sddsMenu
```

APSMenuFrameMake tclVarName

```
Builds a menu or submenu specification. Takes a tcl variable name and adds menu
specification data to that variable each time the procedure is called. Each
invocation adds a new label or entry to the menu specification. Several menu
entry types are available: label, exec, secureExec, command, menu.
Options:
-type <string>          label, exec, secureExec, command, or menu
-text <string>          text for label or menu button
-executable <string>    name of executable if type == exec or secureExec
-contextHelp <string>   context help for that menu entry
-allowedUsers <string>  list of legal users if type == secureExec
-menuList <list>        submenu specification if type == menu
-command <string>       tcl command script if type == command
```

```
Example: (see APSMenuFrame)
# Note: for -executable, just supply the executable, do not suffix with &
# Note: for -text, do not add "..." to button names. This is done for you.
```

APSLabeledEntryFrame widget

```
Creates a titled frame containing a collection of vertically (or horizontally)
stacked entry widgets. Text entered in the widget will set the associated
variable in -variableList.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-label <string> frame title
-variableList <list>
-width <string> width of entry widget
-orientation <string> where <string> is horizontal or vertical (default)
-contextHelp <string>
Creates:
    $parent$widget.
    label
    frame.entry1
    frame.entry2,...,frame.entry<n>
```

APSLabeledOutputFrame widget

```
Creates a titled frame containing a collection of vertically (or horizontally)
stacked text output widgets. Text displayed is taken from the associated variable
in -variableList.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-label <string> frame title
-variableList <list>
-width <string> width of entry widget
-orientation <string> where <string> is horizontal or vertical (default)
-contextHelp <string>
Creates:
    $parent$widget.
    label
    frame.entry1
    frame.entry2,...,frame.entry<n>
```

APSScrolledText widget

```
Creates a text widget with a scrollbar to the right.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-width <string>
-height <string>
-name <string>
-contextHelp <string>
Creates:
    $parent$widget.
    text
    scroll

Example:
APSScrolledText .stuff -parent .userFrame
.userFrame.stuff.text insert end "Add this text to scrolled text widget."
```

APSScrolledList widget

```
Creates a listbox widget with a scrollbar to the right. If a callback procedure
is given with the -callback option, the procedure will be invoked each time the
user single and double clicks on the listbox item. If -itemList option is used,
the width is set to display the widest item in given list.
Options:
-parent <widget>
```

```
-noPack 1
-packOption <list>
-height <string>
-name <string>
-itemList <list> Put these strings on scrolled list
-callback <procedure> note: procedure must be <proc> listboxItem doubleClick
-contextHelp <string>
Creates:
    $parent$widget.
    listbox
    scroll

Example:
proc myCallback {listboxItem doubleClick} {
  if {$doubleClick == 0} {
    puts "you single clicked on $listboxItem"
  } else {
    puts "you double clicked on $listboxItem"
  }
}
APSScrolledList .list -parent .userFrame -callback myCallback
set listbox .userFrame.list.listbox
# Note: items added below may instead be supplied via -itemList option
$listbox insert end "A new entry"
$listbox insert end "Another new entry"
```

APSScroll widget

Creates a vertially scrolled canvas into which you may pack arbitrary widgets. After packing in the desired widgets, you must call APSScrollAdjust to set up the scrolling parameters.

```
Options:
-parent <widget>
-noPack 1
-packOption <list>
-name <string> name for window if you don't supply -parent
-contextHelp <string>
Creates:
    $parent$widget.
    frame.canvas
    frame.yscroll
    frame.canvas.frame
Returns:
    $parent$widget.frame.canvas.frame

Example:
set frame [APSScroll .sw -parent .userFrame]
set doodad "A text message"
foreach widget {one two three four five six} {
    APSLabeledOutput .$widget -parent $frame -label Item -textVariable doodad
}
APSScrollAdjust .userFrame.sw -numVisible 3
```

APSScrollAdjust widget

Given the base widget created by APSScroll ($parent$widget), this procedure sets up the scrolling parameters to something reasonable. In general, it is assumed you have packed homogenous widgets, in which case -numVisible determines how many you will see at a time. Scrolling is set up to jump in increments of one widget. If you pack in widgets of differing sizes, or one really large widget, you will have to set -scrollIncrement to suit yourself.

```
Options:
-numVisible <string>
-scrollIncrement <string>
```

APSScrolledStatus widget

Creates a short, vertically scrolled text box with the label Status. **Each new value of -textVariable is recorded in the text box.**

```
Options:
-parent <widget>
-noPack 1
-packOption <list>
```

```
-name <string> name for window if you don't supply -parent
-textVariable <string>
-width <string>
-height <string>
-contextHelp <string>
Creates:
    $parent$widget.
    frame.text
    frame.scroll

Example:
set status "Ready..."
APSScrolledStatus .status -parent .userFrame -textVariable status
set status "Press go to continue"
```

APSRadioButtonFrame widget

```
Creates a titled frame containing a collection of vertically stacked radio
buttons. The programmer specifies the button titles, associated variable and
assigned values via lists.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-label <string>
-variable <string>
-buttonList <list>
-commandList <list>     commands to be executed when respective btn pushed
-valueList <list>
-orientation <string> where <string> is horizontal or vertical (default)
-contextHelp <string>
Creates:
    $parent$widget.
    label
    frame.button1
    frame.button2,...,button<n>

Example:
set num 0
APSRadioButtonFrame .rb -parent .userFrame -label "Select One" -variable num \
  -buttonList {One Two Three} -valueList {1 2 3} -contextHelp \
  "Select a button and the variable will be assigned the corresponding value." \
  -commandList {{puts "One pushed"} {} {puts "Three pushed"}}
```

APSCheckButtonFrame widget

```
Creates a titled frame containing a collection of vertically stacked check
buttons. The programmer specifies the button titles and associated variables. The
variables are set to 1 or 0, depending on the checkbox state.
Options:
-parent <widget>
-noPack 1
-packOption <list>
-label <string>
-buttonList <list>
-commandList <list>     commands to be executed when respective btn pushed
-variableList <list>
-orientation <string> where <string> is horizontal or vertical (default)
-allNone 1      Adds two buttons which select and clear all check buttons
-contextHelp <string>
Creates:
    $parent$widget.
    label
    frame.button1
    frame.button2,...,button<n>

Example:
set cb1 0
set cb2 1
APSCheckButtonFrame .cb -parent .userFrame -label Configure: \
  -buttonList {check1 check2} -variableList {cb1 cb2}
```

2.2.4. SDDS Widgets

These widgets typically allow graphical selection and manipulation of SDDS files.

3. Tcl Library

Detailed descriptions of each procedure are given here, along with an example. Every procedure in this library begins with upper-case **APS**, every global variable utililized by this library begins with lower-case **aps**.

3.1. Procedures

The tcl procedures are broken up into two categories:

- Utility
- SDDS

3.1.1. Utility

APSParseArguments <list>

```
Provides a non-positional, optional argument capability to Tcl/Tk procedures.
This procedure is utilized by the APS Tk widget library. APSParseArguments parses
the list of options in the args variable of the current scope, and creates a
corresponding set of variables containing the option values.
```

- <list>
  A list of option keywords that are accepted by the procedure requesting the parsing. The parsing scans the args list of the calling procedure. Only options from the keyword list will be processed; others are left in the args list. The effect of APSParseArguments is to translate a sequence like "-keyword value" into "set keyword value"; that is, the keyword names are variable names in the calling procedure.

```
Example:
proc APSWhatever {widget args} {
    # First provide default values for the options
    set buttonLabel "NoLabel"
    set buttonCommand ""
    # This call searches "args" for the options given in <list>
    APSParseArguments {buttonLabel buttonCommand}

    button $widget -text $buttonLabel -command $buttonCommand
}

APSWhatever .mybutton -buttonCommand "puts hello" -buttonLabel "Press Me"
```

APSStrictParseArguments <list>

```
Same as APSParseArguments, except: This call will print an error message to
stderr if you supply arguments to the procedure which are not in <list>.
```

APSExec

```
Options:
-unixCommand <string>
-callback <script>
-outputVariable <variable>

Executes the command given by -unixCommand option without blocking the calling
application. If -callback is given, <script> will be executed upon completion of
unixCommand. Output of unixCommand is normally thrown away. If -outputVariable is
given, command output will be stored in the designated global variable.
```

APSSound

```
Options:
-type <string> where <string> is working, alert, or emergency
-volume <string> where <string> is 1 to 100
-iterations <string>
-period <string> where <string> is in milliseconds or "continuous"

Plays a preset soundfile based on given -type. Procedure schedules work and
returns immediately, so iterations are done in background.
```

APSUniqueName <prefix>

```
Returns a unique string which begins with supplied <prefix> string. Useful for
generating widget names automatically.
```

APSUniqueNumber

```
Returns a unique integer string.
```

3.1.2. SDDS

APSGetSDDSColumn

```
Options:
[-fileName <string>]
[-column <string>]
[-page <string>]

Extracts column from sdds file and returns it as a tcl list.
```

APSGetSDDSParameter

```
Options
[-fileName <string>]
[-parameter <string>]
[-page <string>]

Extracts parameter from sdds file and returns it.
```

APSGetSDDSNames

```
Options:
[-fileName <string>]
[-class <string>] where <string> is column (default), parameter, or array.

Extracts data names by class from an sdds file and returns it as a tcl list.
```

APSCheckSDDSFile

```
Options:
[-fileName <string>]

Verify that given fileName is an SDDS1 file.
Returns 1 if true, 0 otherwise.
```

4. Interpreter Extensions

Please see the external document on “et_wish” for information on the tclCa extension.