E U R O P E A N S O U T H E R N O B S E R V A T O R Y

Organisation Européenne pour des Recherches Astronomiques dans l'Hémisphère Austral

Europäische Organisation für astronomische Forschung in der südlichen Hemisphäre

VERY LARGE TELESCOPE

Data Management Division

Tcl and C++ Utilities Package

Programmer's Manual

Doc.No. VLT-MAN-ESO-19400-1550

Issue 1.2

Date 5/16/99

	Name	Date	Signature
Prepared	A. Brighton	5/16/99	
Approved	M. Albrecht		
Released	P. Quinn		

Change Record

Issue/Rev.	Date	Section/Page affected	Reason/Initiation/Document/Remarks
1.0	**11/01/98**	**All**	**Created**
1.2	**11/09/98**	**All**	**Updated, setup HTML and cross refs**

Table of Contents

1 Introduction

In any software development, there is always a certain amount of code that is generic and could be reused in other projects. This manual describes a collection of general purpose, reusable Tcl and C++ utilities. These are C++ classes and routines, Itcl classes and widgets, and support for writing configure scripts in other packages. Most of these were originally part of other packages and have been collected here in order to make them easily available for use by other packages. Among other things, the *tclutil* package, which can be dynamically loaded as a Tcl package, defines error handling used by C++ classes and base classes for C++ and Itcl widgets. In addition, a general purpose widget library is included, as well as C++ support for HTTP and mmap. The package configure script also generates a shell script that can be used by other packages to simplify their own configure scripts.

1.1 Purpose

The purpose of this manual is to describe the *Tcl and C++ utilities* package and its implementation.

1.2 Scope

This document is primarily aimed at software developers using the package. This package is currently used by the *astrotcl*, *rtd*, *cat*, and *skycat* packages.

1.3 Applicable Documents

This document is based on the following documents:

[1] VLT-PRO-ESO-10000-0228, 1.0 10/03/93 -- VLT Software Programming Standards

1.4 Reference Documents

The following documents are referenced in this document:

2 Overview

The *tclutil* package is designed for use in a Tcl/C/C++ environment. It is made up of the following parts:

- A collection of general purpose C++ classes and routines
- A collection of C++ classes that support Tcl/Tk development
- A library of Itcl classes and widgets

In addition, the configure script for the package exports a shell script file, *tclutilConfig.sh*, that can be used by the configure scripts in other packages to save work. The file sets shell variables based on information discovered by configure, in the same way as the tclConfig.sh script provided by Tcl. This tells you, for example, how to make shared libraries on a given system, the locations of various Tcl/Tk packages, the compiler name and options, etc.

The following tables give an overview of the classes and widgets found in this package. For more details, see the following section and the man pages in the Reference section.

General Purpose C++ Classes and Routines

Name	Description
Error Handling	Collection of C++ routines used to report errors. There are different versions for general errors, system errors and warnings. Some of the routines support a printf style interface. You can also supply a user defined error handler to be called for each error.
class HTTP	This is a generic class that can be used to do an HTTP GET on a given URL and return the results.
class Mem	This class makes it easy to memory map a file and use it as a string or other object in memory. SysV shared memory, mmap and malloced memory are all supported by the class in a transparent way.
Other C++ Utilities	This is a collection of general purpose C++ routines, mostly file and string operations.

C++ Classes for Tcl/Tk Development

Name	Description
TclCommand	Base class for C++ classes implementing a Tcl command (widgets and image types are also Tcl commands and are subclasses of this class).
TkWidget	Base class of C++ classes implementing a Tk image type.
TkImage	Base class of C++ classes implementing a Tk widget.

Itcl Classes and Widgets

Name	Description
Batch	Class for evaluatimg Tcl commands in the background.

Itcl Classes and Widgets

Name	Description
ButtonFrame	Widget for a frame with a row of buttons.
CanvasDraw	Line graphic editor for a Tk canvas, supports interactive drawing, moving and resizing of various shapes, in a selected outline and fill color, width, etc.
CanvasPrint	Popup window to print the contents of a Tk canvas (Printing of embedded images is not supported in this version).
CanvasWidget	A canvas frame with optional scrollbars.
CheckEntry	A checkbutton and an entry widget in a frame.
ChoiceDialog	Popup window with a title and radiobuttons, to get the user to make a choice.
Chooser	Simple popup widget for selecting items based on filename, suffix and directory.
DialogWidget	Base class for popup dialog widgets.
DoubleList	Frame displaying 2 listboxes with arrows in between.
DoubleTableList	Frame with 2 TableList widgets and arrows in between.
EntryForm	Popup window with a configurable list of labels with entries.
FileSelect	Popup motif style file select dialog.
FrameWidget	Base class for widgets contained in a frame.
GraphPrint	Popup dialog to print a BLT graph.
InputDialog	Popup dialog to get input from the user.
LabelCheck	Frame containing a label and a checkbutton.
LabelChoice	Frame containing a label and a list of radiobuttons.
LabelEntry	Frame containing a label and an entry.
LabelEntryScale	Frame displaying a label, an entry and a scale widget (type in value or use slider).
LabelMenu	Frame displaying a label and a menubutton.
LabelMessage	Frame displaying a label and a message (for read-only items).
LabelNumber	Frame displaying a label and a number.
LabelValue	Frame displaying a label and some value.
LabelWidget	Base class of widgets with a label.
ListDialog	Dialog to display a listbox and some text in a popup window.
ListboxWidget	Listbox frame with optional scrollbars, methods to set/get selection and contents.
PrintDialog	Base class of print dialogs.
ProgressBar	Frame containing a netscape style progress bar and message.
TableList	Listbox based table widget with headings and columns formatted automatically, supports sorting, rearranging columns, printing,

Itcl Classes and Widgets

Name	Description
TableListConfig	Popup widget for editing the table layout for the TableList class.
TableListConfigFile	Itcl class used to save configuration information for the TableList class.
TableListPrint	Print dialog for the TableList class.
TableListSort	Popup window used to set sorting options for the TableList class.
TextDialog	Popup dialog displaying a text window and a specified text.
TopLevelWidget	Base class of widgets with a top level window. Also provides methods for adding a menubar with menus and menu items at top and a short help window at the bottom. Provides support for user defined *plugins* for any subclass.

3 User's Guide

This section describes in general the contents of the *tclutil* package. For details, see the man pages in the *Reference* section.

3.1 General Purpose C++ Classes and Routines

3.1.1 Error Handling

Error handling in the C++ code consists of a collection library routines, described in *error(3)*, and a facility for specifying an error handler routine to be called to report errors. The error reporting routines keep a copy of the most recent error message, which can be retrieved using the `getError`() routine. In Tcl based applications, these routines normally generate Tcl error messages in the main interpreter, while in non-Tcl applications, the default action is to print the error messages on stderr. The *TclCommand(3)* class automatically defines an error handler for Tcl.

C++ Error Reporting Routines

Routine	Description
`error`	Report the error.
`fmt_error`	Same as `error`, but with a *printf* like interface.
`sys_error`	Report the error with the system error code, like perror.
`fmt_sys_error`	same as `sys_error`, but with a *printf* like interface.
`log_message`	Used to log a message that is not an error (not used here).
`last_error`	Return the text of the previous error message.
`last_error_code`	Return the error code for the previous error.
`clear_error`	Reset the last_error buf to empty.
`set_error_handler`	Set a routine to be called when errors occur and return a pointer to the previous handler.
`print_error`	Simple default error handler: print the error message on stdout.

3.1.2 class HTTP

This class is used to access data over the network using HTTP GETs. Methods are available to get the data, given the URL and to step through the data line by line or return the entire result in a buffer. The syntax of the URL may be one of the following:

- `http`*://host:port/path* (:*port* is optional)
- `file`*://path*
- *command args*

In the first case, an HTTP GET is done to retrieve the data for the URL. If the URL starts with "`file`:", the contents of the local file is returned. If the URL does not start with "`file`:" or "`http`:",

it may be a local command to execute. In this case, the command is executed and the output of the command is returned. This last case is only allowed if explicitly enabled with the `allowUrlExec`() method. This should only be done if you know that it is safe to execute the command.

The following example does an HTTP get on *url* and prints the result:

```
HTTP http;
int nlines = 0;
char* buf = http.get(url, nlines);
printf("answer = (%d lines)\n%s\n", nlines, (buf ? buf : "ERROR"));
```

See the man page *HTTP(3)* for details on the various constructors and methods.

3.1.3 class Mem

This class uses reference counting to help manage memory areas. The memory may be allocated with the C++ "`new`" operator, SysV shared memory or mmap. The class keeps track of who owns the memory, who is responsible for deleting it when no longer needed and how many references there are to the memory area. When there are no more references, we can safely detach a shared memory area and if we are the "owner", delete it. Constructor arguments and flags determine whether shared memory, mmap or normal (unshared) memory is allocated or whether existing shared memory is used. The state can be changed at any time, i.e.: shared memory can be changed to unshared and unshared to shared.

The following example allocates 10000 bytes of sysV shared memory:

```
Mem m(10000, 1);
char* p = (char*)m.ptr();
```

This example uses mmap to map a file to memory:

```
Mem m(filename);
char* p = (char*)m.ptr();
```

See the man page *Mem(3)* for details on the various constructors and methods.

3.1.4 Other C++ Utilities

The file util.C contains a number of utility routines for string and file manipulation[1]. This includes routines to strip blanks from a string, get the size of a file, get the base name of a file or the real name (follow a symbolic link), unbuffered reading and writing to a socket, etc. See *util(3)* for a description.

3.1.5 Test Cases

The `util/test` subdirectory contains test cases for the C++ classes. Type "`make test`" to run the tests. The results are compared with a previous run and an error message is printed if something changed.

1. These routines should probably be encapsulated in a class or use a common prefix, but they are kept for now for convenience and to remain compatible with existing software.

3.2 C++ Support for Implementing Tcl Commands, Widgets and Image Types

3.2.1 TclCommand

`TclCommand` is a base class for C++ classes defining Tcl commands. It offers support for an [incr Tk] like interface. That is, Tcl commands defined in this framework can be declared as Tcl objects, where each object may have any number of methods or subcommands. Tcl subcommands are defined in a static array mapping the name of the subcommand to the C++ method that implements it. Inheritance is supported so that new subcommands may be added in a derived class. Each class in the hierarchy defines the virtual *call* method, which calls a C++ method, given its name and passes control to the base class if it does not know the method. The methods that implement the Tcl subcommands all take the same arguments: the argc and argv arguments that were passed to the command (minus the command name).

See *TclCommand(3)* for more details and examples.

3.2.2 TkWidget

TkWidget is a base class for C++ classes defining Tk widgets. In addition to the features inherited from its base class *TclCommand(3)*, `TkWidget` adds support for creating and configuring Tk widgets.

The configure (or config) and cget subcommands are implemented here automatically, so that derived classes don't have to handle this. The derived classes only need to define two structures required by Tk for widgets:

One is the `Tk_ConfigSpec` structure, a static struct describing the widget options.

The other is a non-static member struct for holding the values for widget configuration options. This struct must be derived from `TkWidgetOptions`, which is just a dummy struct used for passing the actual struct as a parameter.

See below and *TkWidget(3)* for more details.

3.2.3 TkImage

`TkImage` is a base class for a class defining a new Tk image type. The image type defined here differs slightly from standard Tk image types, such as photo or bitmap in that it is only allowed to display a given image in a single widget and you can optionally specify what type of widget that should be.

The intent here is that you specify that the image can only be in, say, a canvas window. Since a given image "instance" can only be displayed in one canvas window (as far as Tk is concerned), you can use the scrolling offsets of the canvas to implement intelligent scrolling. It is still possible to display an image in multiple widgets, but this must be handled in the derived class (see `RtdImage(3)` for an example). The restriction comes about, because of the way Tk handles images in multiple widgets. Normally, if one changes size, the others do as well. This is not necessarily what you always want in an image processing application. You may want to have the same image at different sizes sharing the same data.

This class does some of the abstract work for dealing with Tk images. It declares some of the static member functions that Tk calls for image handling (`GetImage`, `DisplayImage`, `FreeImage` and `DeleteImage`) and calls virtual member functions where necessary to handle the actual display of the image.

This class also defines and implements the "`configure`" and "`cget`" image commands. The derived class passes the `TkImage` constructor information about the configuration options (`optionsPtr_` and `configSpecsPtr_`) for use in these subcommands. The derived class may still need to redefine the "`configureImage`" method in order to handle certain options, but most of the work is done in this base class. See *TkWidget(3)* for more details.

3.2.4 bitmaps

The *tclutil/bitmaps* directory contains the bitmaps used by the widgets in this package. In order to make it easier to use the bitmaps and to release a single binary of applications, using packages such as ET (Embedded Tk), the bitmaps are compiled in as static bitmaps. They are available to all Tcl scripts by name (the base name of the bitmap file). Other packages can add their own bitmaps in the same way by compiling them as static bitmaps. This adds very little overhead and saves a lot of trouble locating the bitmaps at runtime. It may also increase Tk performance somewhat.

3.2.5 TclUtil

The *tclutil/src/TclUtil.C* file defines the code necessary to make this package loadable in Tcl. When configured with the `--enable-shared` option, a single shared library file is generated (libtclutil.so, or libtclutil.sl). This library may be loaded dynamically in a Tcl application using the Tcl "`package require`" command. In order for the package to be located, the global variable `auto_path` must include the directory where the tclutil package is installed. See the Tcl documentation for the `package(n)` command for more details.

3.3 Itcl Classes and Itk Widgets

This section gives an overview of the available Itcl widget classes. For details, methods and options, see the man pages in the Reference section.

3.3.1 Batch

This class is used to evaluate a set of Tcl commands in a separate process, so that it can be interrupted. Using the `bg_eval` method, you specify a command to evaluate in the background. When it is done, a callback method is called with the status and the results of the command. No exec is done, so the process is just a copy of the current one and has a copy of the memory and open files. This class is often used together with the *ProgressBar(n)* class, to do some processing in the background and allow the user to interrupt it.

This class does not have any visible window, although it does define a frame, which is used to make sure the object is automatically deleted when the class using it is deleted.

The following example evaluates the Tcl command `get_preview` in a separate process and calls `preview_done` with the results:

```
Batch $w_.batch \
    -command [code $this preview_done] \
    -debug $itk_option(-debug)
$w_.batch bg_eval [code $this get_preview]
```

3.3.2 ButtonFrame

This class is an itcl widget for displaying a frame with a row of standard buttons, such as *OK* and *Cancel*, with options for adding and other buttons and actions.

The example below calls the *ok* method if OK is pressed and the *quit* method if quit is pressed:

```
set w [ButtonFrame $w_.b \
   -ok_cmd [code $this ok] \
   -cancel_cmd [code $this quit]]
pack $w -fill x -expand 1
```

See *ButtonFrame(n)* for details.

3.3.3 CanvasDraw

`CanvasDraw` is an Itcl widget class that an application can use to add interactive drawing capabilities to any Tk canvas. Since it is a subclass of `TopLevelWidget`, it creates its own toplevel window containing a drawing mode selection section and an option section for setting colors, fonts, line thickness, arrow types, etc. The window can be hidden on startup by specifying the "-withdraw 1" option inherited from `TopLevelWidget`. The widget provides methods to add a *Graphics* menu to another top level window (shown above at right). You can also load an image for the background of the canvas and draw graphics over the image. Methods are available to limit the drawing to the size of the image. All of the graphic objects may be moved or resized interactively by the user using FrameMaker style *handles*.

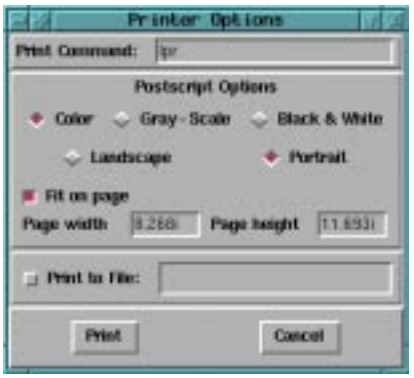
Here is the code for the example shown above (The menu code is not shown):

```
class tCanvasDraw {
    inherit TopLevelWidget

    constructor {args} {
        add_menubar
        itk_component add canvas {
            CanvasWidget $w_.canvas \
               -canvasbackground black
        }
        pack $itk_component(canvas) -fill both -expand 1
        set canvas [$itk_component(canvas) component canvas]

        itk_component add draw {
```

```
            CanvasDraw $w_.draw\
               -canvas $canvas \
               -withdraw 1 \
               -transient 1 \
               -center 0
         }
         add_menu_items
         set image [image create photo -file mickey.gif]
         $canvas create image 40 40 -image $image -anchor nw
         eval itk_initialize $args
      }
   }
```

See *CanvasDraw(n)* for more information.

3.3.4 CanvasPrint



This class extends the PrintDialog class to be able to print the contents of a canvas as postscript. There are options to print in color, grayscale or mono, set the page size and orientation. You can print to a file or directly to a postscript printer.

Example:

```
CanvasPrint $w_.print
```

See *CanvasPrint(n)* for more details.

3.3.5 CanvasWidget

This is an Itcl class based on the Tk canvas widget. It creates a frame containing a canvas window and optional scrollbars.

Example:

```
set w [CanvasWidget .c]
pack $w -fill x -expand 1
[$w component canvas] create text 100 100 -text “CanvasWidget”
```

See *CanvasWidget(n)* for more details.

3.3.6 CheckEntry

This class defines an itcl widget combining a checkbutton and an entry. This can be used to turn options on and off in a dialog window.

Example:

```
set w [CheckEntry .ce -text “Test label:”]
pack $w -fill x -expand 1
```

See *CheckEntry(n)* for more details.

3.3.7 ChoiceDialog

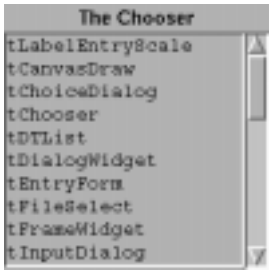
A `ChoiceDialog` is a dialog widget used to display a message and a list of choices to get a choice from the user. A Tcl command is evaluated with the user’s selection when the OK button is pressed.

Example:

```
set w [ChoiceDialog .d \
   -text {Please choose...} \
   -cols 2 \
   -messagewidth 3i \
   -choice {One Two Three Four Five Six Seven Eight} \
   -value {One}]
puts “Dialog returned: [$w activate]”
```

See *ChoiceDialog(n)* for more details.

3.3.8 Chooser



`Chooser` is a simple itcl widget for selecting items based on files with a given suffix in a given directory.

Example:

```
set w [Chooser .c \
   -title “The Chooser” \
   -dir . \
   -suffix .tcl \
   -files [glob ./*.tcl] \
   -command puts]
```

See *Chooser(n)* for more details.

3.3.9 DialogWidget

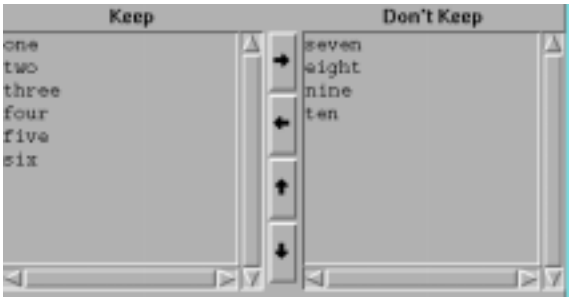
A DialogWidget is the base class of dialog widgets. It is defined here as a window with a message at top and a row of buttons at bottom. Through inheritance, you can add widgets inbetween.

Example:

```
set w [DialogWidget .d \
   -text {This is a test message} \
   -bitmap warning \
   -default 1 \
   -buttons {OK Cancel Help}]
```

See *DialogWidget(n)* for more details.

3.3.10 DoubleList



A `DoubleList` is an Itcl widget for displaying two `ListboxWidgets` with arrows between them for moving items back and forth or up and down. This widget is useful for selecting items from a long list of available items. The left and right listboxes are instances of the *ListboxWidget(n)* class and have the Itcl component names *left* and *right*.

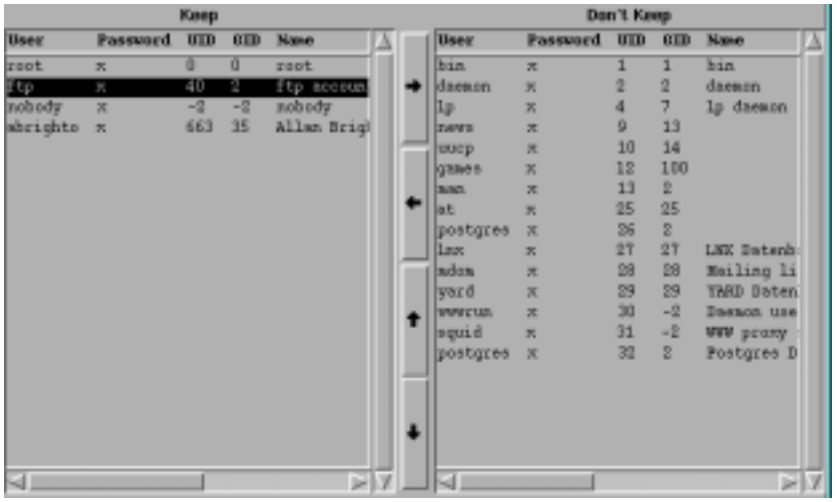
Example:

```
itk_component add dlist {
    DoubleList $w_.dtlist \
    -lefttitle “Keep” \
    -righttitle “Don’t Keep” \
    -hscroll 1 \
    -updown 1 \
    -selectmode extended
}
```

```
pack $itk_component(dlist) -fill both -expand 1

[$itk_component(dlist) component left] set_contents \
    {one two three four five six}
[$itk_component(dlist) component right] set_contents \
    {seven eight nine ten}
```

See *DoubleList(n)* for more details.

3.3.11 DoubleTableList



A `DoubleTableList` is an Itcl widget for displaying two `TableLists` with arrows between them for moving items back and forth or up and down. You can access the left and right `TableLists` as itk components *left* and *right*. This widget is used in the same way as the `DoubleList` widget, but is better suited for tabular data that should be formatted in columns. The *left* and *right* components are instances of the *TableList(n)* widget class.

Example:

```
# create the DoubleTableList
itk_component add dtlist {
    DoubleTableList $w_.dtlist \
        -lefttitle “Keep” \
        -righttitle “Don’t Keep” \
        -hscroll 1 \
        -updown 1 \
        -selectmode extended
}
pack $itk_component(dtlist) -fill both -expand 1
set headings {User Password UID GID Name Home Shell}
set info {}
set fd [open /etc/passwd]
while {[gets $fd buf] != -1} {
    lappend info [split $buf :]
}

$itk_component(dtlist) config \
    -headings $headings \
    -leftinfo $info \
    -rightinfo {}
```

See *DoubleTableList(n)* for more details.

3.3.12 EntryForm

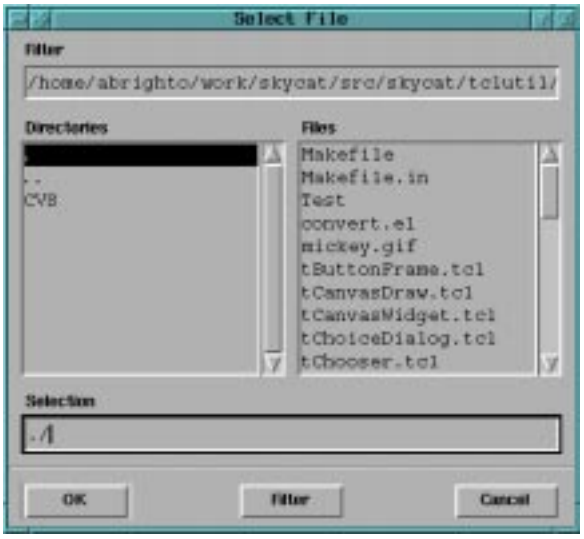
An `EntryForm` widget is a dialog for entering data at given labels. The title and labels are specified as options.

Example:

```
EntryForm .ef \
    -title {Please enter the data:} \
    -labels {Name Address Tel Fax Age Height Weight} \
    -command puts \
    -values {{My Name} {My Address} {My Tel} {My Fax} {My Age} \
            {My Height} {My Weight}}
```

See *EntryForm(n)* for more details.

3.3.13 FileSelect



This class implements a version of the OSF/Motif style file selection dialog box. This is a modified version of an older widget written by Mark Ulferts (before Iwidgets was released) that is still used because of poor performance in the iwidgets2.x version. This should probably be replaced with the new Tk version when we move to tcl8.x (Tk 8.x includes a standard file selection widget).

Example:

```
puts “got: [filename_dialog]”
```

See *FileSelect(n)* for more details.

3.3.14 FrameWidget

The `FrameWidget` Itcl class is the base class of widgets that are contained in a Tk frame (see also *TopLevelWidget(n)* for widgets that have thier own top level window). `FrameWidget` is a subclass of `itk::Widget` and thus inherits all of the features described in *Widget(n)*. In addition, a number of useful methods are defined, for use by the derived classes.

Both the `FrameWidget` and `TopLevelWidget` classes call the `init` method when all the constructors have completed (using the `after idle` command). The `init` method is often the best place to build widgets, since at that point you know the values of all of the Itcl options.

Example:

```
class util::CanvasWidget {
    inherit util::FrameWidget
    ...
}
```

See *FrameWidget(n)* for more details.

3.3.15 GraphPrint

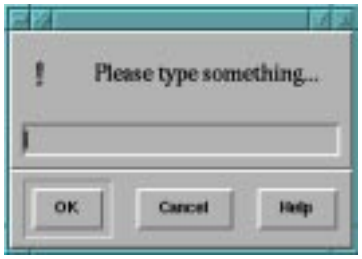
This widget defines a print dialog box for printing the contents of a BLT graph. This widget looks and acts just like *CanvasPrint(n)*, except that it works with a BLT graph rather than a Tk canvas.

Example:

```
GraphPrint $w_.print
```

See *GraphPrint(n)* for more details.

3.3.16 InputDialog



An `InputDialog` is a dialog to display a message and get input from the user.

Example:

```
set w [InputDialog .d \
   -text {Please type something...} \
   -bitmap warning \
   -modal 1 \
   -buttons {OK Cancel Help}]
puts "Dialog returned: [$w activate]"
```

See *InputDialog(n)* for more details.

3.3.17 LabelCheck

`LabelCheck` is an Itcl megawidget for choosing options using a label and radiobuttons (uses blt::table to arrange the buttons in rows and columns).

Example:

```
set w [LabelCheck .lc \
  -text {Test Label:} \
  -rows 3 \
  -choice {one two three four five six seven eight nine ten} \
  -value {three six nine} \
  -command puts]
pack $w -fill x -expand 1
```

See *LabelCheck(n)* for more details.

3.3.18 LabelChoice

`LabelChoice` is an Itcl megawidget for choosing options using a label and a set of radiobuttons. The widget uses `blt::table` to arrange the buttons in rows and columns. You can specify the number of rows or columns and a list of items as options. There are methods to get a list of selected values or you can specify a command to be evaluated whenever the selected values change.

Example:

```
set w [LabelChoice .lc \
    -orient vertical \
    -text “Test label:” \
    -choice {one two three four five six seven eight \
            nine ten eleven twelve {a b c} {d e f}} \
    -value three \
    -relief groove \
    -anchor w \
    -borderwidth 3 \
    -command puts]
pack $w -fill both -expand 1
```

See *LabelChoice(n)* for more details.

3.3.19 LabelEntry

This widget displays a label and an entry and implements convenient methods for accessing and modifying the label and the value.

Example:

```
set w [LabelEntry .le \
    -text “Test label:” \
    -value 42 \
    -labelwidth 20 \
    -valuefont 8x13 \
    -relief sunken \
    -borderwidth 3 \
    -command puts \
    -textvariable “myvar”]
pack $w -fill x -expand 1
```

See *LabelEntry(n)* for more details.

3.3.20 LabelEntryScale

`LabelEntryScale` is an Itcl widget combining a labeled entry with a scale widget to make a scale with an editable entry field. This widget is useful for enterring numerical values.

Example:

```
set w [LabelEntryScale .les \
    -text “Test label:” \
    -from 10 \
    -to 120 \
    -value 25 \
    -valuewidth 4 \
    -validate real \
    -relief sunken \
    -borderwidth 3 \
    -command puts \
    -entrycommand foo \
    -validate real \
    -orient horizontal]
```

See *LabelEntryScale(n)* for more details.

3.3.21 LabelMenu

This widget displays a label and a menubutton with a selector and a menu of radiobuttons. This can be used for choosing items from a list and displaying the current choice. The widget supports adding of items to the radiobutton menu and keeps track of which items are selected.

Example:

```
set w [LabelMenu .lm \
    -text “Test label:” \
    -valuefont 8x13 \
```

```
        -relief raised \
        -orient horizontal]
    pack $w -fill x -expand 1
    $w add -label “one” -command “foo one”
    $w add -label “two” -command “foo two”
    $w add -label “three” -command “foo three”
```

See *LabelMenu(n)* for more details.

3.3.22 LabelMessage

`LabelMessage` is an Itcl widget for displaying a label and a (read-only) message.

Example:

```
    set w [LabelMessage .lm \
        -text “Test label:” \
        -value “Some Message...” \
        -labelwidth 10 \
        -labelfont -Adobe-Helvetica-Bold-R-Normal--*-140-*-*-*-*-*-* \
        -valuefont 8x13 \
        -relief sunken \
        -borderwidth 3 \
        -aspect 1000 \
        -orient horizontal]
    pack $w -fill x -expand 1
```

See *LabelMessage(n)* for more details.

3.3.23 LabelNumber

`LabelNumber` is an Itcl widget for displaying a label, a number and buttons to increment and decrement the number.

Example:

```
    set w [LabelNumber .ln \
        -text “Test label:” \
        -value 42 \
        -labelwidth 20 \
        -labelfont -Adobe-Helvetica-Bold-R-Normal--*-140-*-*-*-*-*-* \
        -valuefont 8x13 \
        -relief sunken \
        -borderwidth 3 \
        -command puts \
        -orient horizontal]
    pack $w -fill x -expand 1
```

See *LabelNumber(n)* for more details.

3.3.24 LabelValue

This widget displays a label and a read-only value and implements convenient methods for accessing and modifying the label and the value.

Example:

```
set w [LabelValue .le \
    -text “Test label:” \
    -value “test value”]
pack $w -anchor w -fill x
```

See *LabelValue(n)* for more details.

3.3.25 LabelWidget

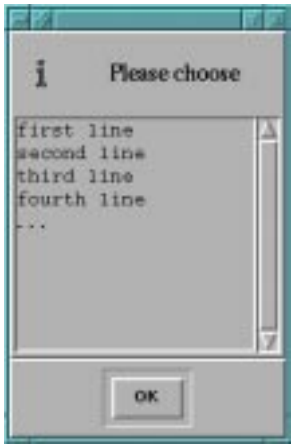
LabelWidget is an Itcl widget for displaying a label and is the base class of labeled widgets. Derived classes add other widgets to the right or under the label.

Example:

```
set w [LabelWidget .le \
   -text “Test label:” \
   -labelwidth 20]
```

See *LabelWidget(n)* for more details.

3.3.26 ListDialog



A `ListDialog` is a dialog to display a listbox with a number of choices. When the user makes a selection and types OK or <Enter>, the widget returns the value via the activate method.

Example:

```
set w [ListDialog .d \
      -text {Please choose} \
       -choice {{first line} {second line} {third line} {fourth line} {...}} \
      -modal 0]
puts “Dialog returned: [$w activate]”
```

See *ListDialog(n)* for more details.

3.3.27 ListboxWidget

`ListboxWidget` is an Itcl widget for scrolled lists, based on the Tk listbox. Methods are available that make it easy to set and get the contents of the listbox.

Example:

```
set w [ListboxWidget .lb \
   -title “The Listbox”]
pack $w -fill x -expand 1

$w set_contents {
    {one   two   three ...}
    {four  five  six ...}
    {seven eight nine ...}
}
$w select_row 1
puts “selected [$w get_selected]”
puts “internal listbox name is: [$w component listbox]”

bind [$w component listbox] \
<ButtonRelease-1> “puts \[$w get_selected\]”
```

See *ListboxWidget(n)* for more details.

3.3.28 PrintDialog

`PrintDialog` is the base class of Popup dialogs for specifying printer options. See *CanvasPrint(n)* for an example of how this is used or see the man page *PrintDialog(n)* for details.

3.3.29 ProgressBar

This object displays a message and a scale indicating that work is in progress. Methods are available to set the message to display and the percent done.

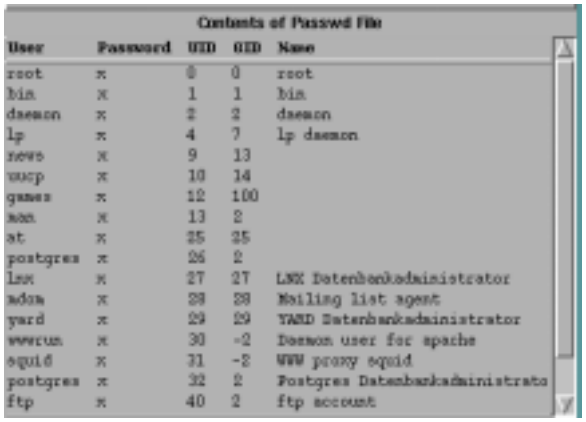
Example:

```
set w [ProgressBar .pb -text “Test label:” -from 0 -to 100 -value [set v 0]]
pack $w -fill x -expand 1

# dummy proc to animate the progress bar
proc foo {} {
    global w v
    incr v
    $w config -text “${v}% done” -value $v
    if {$v >= 100} {
```

```
            exit
        }
        after 100 foo
    }
    after 100 foo
```

See *ProgressBar(n)* for more details.

3.3.30 TableList



`TableList` is an itcl widget for displaying tabular information and headings in a Tk listbox. It lines up columns of data (specified in tcl list format) with column headings, optionally sorts the data by given columns, in a given order, can “hide” columns, rearrange columns, or display columns matching certain expressions. You can even print the contents of the table. This widget has been in use for a number of years and has grown to include quite a few features. It is well suited to displaying the results of database or catalog queries.

Example:

```
# create the TableList
itk_component add tlist {
    TableList $w_.tlist \
        -title “Contents of Passwd File” \
        -selectmode extended
}
pack $itk_component(tlist) -fill both -expand 1
...
set fd [open /etc/passwd]
set info {}
set headings {User Password UID GID Name Home Shell}
while {[gets $fd buf] != -1} {
    lappend info [split $buf :]
}
$itk_component(tlist) config \
    -headings $headings \
    -info $info
```

See *TableList(n)* for more details.

3.3.31 TableListConfig

`TableListConfig` is an Itcl popup widget for editing the table column layout for the *TableList(n)* class. This can be used to specify that certain columns should not be displayed or should be displayed in a different order. Although it is possible to create an instance of this class yourself, it is normally done with the `TableList` method `layout_dialog`.

Example:

```
TableList $tablelist -headings {...}
...
$tablelist layout_dialog
```

See *TableListConfig(n)* for more details.

3.3.32 TableListConfigFile

`TableListConfigFile` is an Itcl class for managing the configuration information for the *TableList(n)* widget. This class does not have a visible window interface. It creates a frame, but only to ensure proper cleanup when the TableList window is deleted. See also *TableListConfig(n)* for the user interface to this class. This class is internal to TableList and should not normally be accessed from the outside.

3.3.33 TableListPrint

`TableListPrint` is a print dialog box for printing the contents of a *TableList(n)*. It is a subclass of *PrintDialog(n)*. You can choose which columns to print by selecting or deselecting the column names. Although you can use the class directly, it is usually easier to just call the `TableList::print_dialog` method.

Example:

```
TableList $tablelist -headings {...}
$tablelist print_dialog
```

See *TableListPrint(n)* for more details.

3.3.34 TableListSort

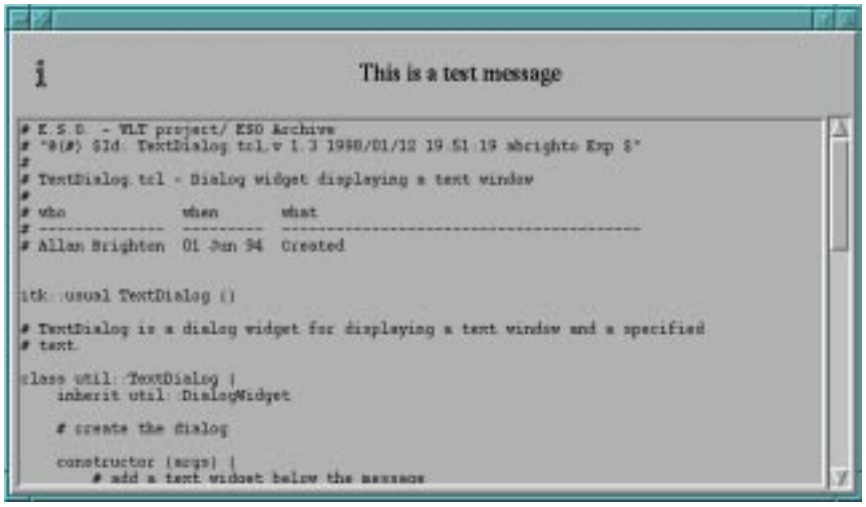
`TableListSort` is an Itcl popup widget for sorting contents of a *TableList(n)*. You can specify one or more columns to sort the list by and the direction of the sort (*increasing* or *decreasing*). Although you can use the class directly, it is normally easier to simply call the `TableList::sort_dialog` method.

Example:

```
TableList $tablelist -headings {...}
$tablelist sort_dialog
```

See *TableListSort(n)* for more details.

3.3.35 TextDialog



`TextDialog` is a dialog widget for displaying a text window and a specified text.

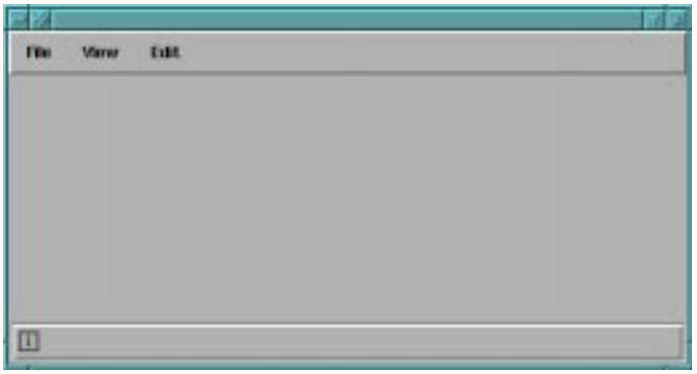
Example:

```
set w [TextDialog .d \
    -text $text\
   -contents $contents]
```

```
$w activate
```

See *TextDialog(n)* for more details.

3.3.36 TopLevelWidget



The `TopLevelWidget` Itcl class is a subclass of `itk::Toplevel` and thus inherits all of the features described in *Toplevel(n)*. In addition, a number of useful methods are defined, for use by the derived classes. These include support for adding a menubar, menu buttons and menu items to the top of the window and adding a short help window at the bottom. A short help text is defined for each menu item and widget and it is displayed (in the short help window) whenever the mouse passes over it.

Both the `FrameWidget` and `TopLevelWidget` classes call the `init` method when all the constructors have completed (using the `after idle` command). The `init` method is often the best place to build widgets, since at that point you know the values of all of the Itcl options.

Any subclass of `TopLevelWidget` automatically has support for a user defined *plugin*. For any class named *Foo*, for example, a Tcl proc named *Foo_plugin* many be defined. The plugin proc is called after the class construction is complete (after calling *init*) with the name of the class instance. If the environment variable *FOO_PLUGIN* is defined (replace *FOO* with the class name in upper case), it is assumed to be the name of the Tcl source file containing the plugin code and defining the Tcl proc *Foo_plugin*. The file is sourced and the file's directory is added to the Tcl `auto_path` variable.

This plugin feature can be used to add features to widgets, such as additional menus or buttons. Once you have the *handle* for the top level widget, it is usually easy to access other internal widgets, if necessary, to make any changes or additions you want. The Tcl language is also very flexible and will allow you to redefine procedures and methods at run time.

The normal way to use the `TopLevelWidget` class is to declare it as a base class:

```
inherit TopLevelWidget
```

However, you could also create an instance directly, for example:

```
TopLevelWidget .t
.t add_menubar
.t add_menubutton File
.t add_menubutton View
.t add_menubutton Edit
.t make_short_help
```

See *TopLevelWidget(n)* for more details.

3.3.37 Test Cases

The *tclutil/test* directory contains test and demo scripts for all of the widgets. To run the tests, cd to the *tclutil/test* directory and type `Test`.

3.3.38 Tcl Utility Procs

This package also defines some simple, yet usefull Tcl utility procedures that are described below.

Utility Procs

utilGetTopLevel	Return the name of the top level window for $w, or an empty string if $w is the root window ".".
utilGetSelection	Get the current selection and return it or return an empty string if there is no selection.
utilRaiseWindow	Make the given top level window visible by raising it, if it is hidden, or deiconifying it if it is not mapped.
utilReUseWidget	If the given widget exists, make it visible and reconfigure it with the given options, otherwise create it with the given options.
utilPrintErrors	For debugging: causes all errors to be printed on stderr in addition to any other handling.

The file *udialog.tcl* also defines a number of utility procedures for common dialog windows.

Dialog Procs

filename_dialog	Choose a file with a file browser.
confirm_dialog	Ask for confirmation before doing something.
error_dialog	Report an error message.
errexit_dialog	Error message dialog with an *Exit* button for serious errors.
warning_dialog	Pop up a window with a warning message.
info_dialog	Display an informational message.
choice_dialog	Ask the user to make a choice form a number of items.
input_dialog	Ask the user to type something in.
action_dialog	Ask the user to do something or press cancel.

4 Reference

This section contains man pages for the shell commands, C++ and Itcl classes and C routines used in this package.

4.1 COMMANDS

4.1.1 itcldoc(1)

NAME

itcldoc - script to extract man pages from itcl class source files

SYNOPSIS

itcldoc ?filenames?

DESCRIPTION

Itcldoc is a tcl script used to extract man pages from itcl class source files. The itcldoc script assumes a certain style of coding, but does not require any special hints otherwise. It assumes comments precede each declaration in comment blocks. The script splits an itcl file into a Tcl list. It first filters the comments to make them list elements. It then looks for declaration keywords in the list: class, method, proc, itk_option, public, protected,... and sorts and documents them based on the comments in the code preceding the declarations.

In addition, each source file should have a comment header containing a line in the format:

filename.tcl - short description

It is assumed that each file given defines one itcl class.

This script is only used for itcl files defining an itcl class. By convention itcl classes start with an upper case letter and the file names for files that contain classes should have the same name with a ".tcl" extension.

See the source files and man pages in this package for examples.

- - - - - -
Last change: 07 May 99

4.2 C++ CLASSES, C ROUTINES

4.2.1 error(3)

NAME

error - C++ routines used for error reporting

SYNOPSIS

```
#include "error.h"

enum {OK, ERROR};

int error(const char* msg1, const char* msg2="", int code = 0);
int sys_error(const char* msg1, const char* msg2="");

int fmt_error(const char* fmt, ...);
int fmt_sys_error(const char* fmt, ...);

char* last_error();
int last_error_code();
void clear_error();

void (*set_error_handler(void (*error_handler)(const char*)))(const char*);
void print_error(const char* msg);

void log_message(const char* fmt, ...);
void (*set_log_message_handler(void (*message_handler)(const char*)))(const char*);
void print_log_message(const char* msg);
```

DESCRIPTION

These C++ routines are used to report errors and log messages. Some of the routines take one or two char strings as arguments to print, others have a "printf" style interface, to make it easier to format the error messages. In some cases the system error description (errno) is appended to the message. The default action is to print the error on stderr, however an error handler may be defined to override this behavior.

An error handler is a C++ routine of the form:

```
void errhandler(const char* msg)
```

where "msg" is the text of the error message, already formatted, if needed.

ROUTINES

error(msg1, msg2, code)
: Concatenate msg1 and the optional msg2 and report the error with the optional system error code.

sys_error(msg1, msg2)
: Concatenate msg1 and the optional msg2 and report the error, including the system error code, in the same way as perror(1).

fmt_error(fmt, ...)
: Report the error with a printf(1) style interface.

fmt_sys_error(fmt, ...)
: Report a system error with a printf style interface.

last_error()
: Return the text of the previous error message.

```
last_error_code()
        Return the error code for the previous error.

clear_error()
        Reset the last_error buf to empty.

set_error_handler(errhandler)
        Set a routine to be called when errors occur and return a
        pointer to the previous handler.

print_error(msg)
        Simple error handler: print the message.

log_message(fmt, ...)
        Print a message (like printf).

set_log_message_handler(msghandler)
        Set a routine to be called for log messages and return a
        pointer to the previous handler.

print_log_message(msg)
        Simple message handler: print the message.
```

SEE ALSO

TclCommand

- - - - - -

Last change: 07 May 99

4.2.2 ErrorHandler(3)

NAME

```
ErrorHandler - A C++ class for managing Tk Error Handlers for
                    catching X errors
```

SYNOPSIS

```
#include "ErrorHandler.h"

/*
 * This class is used to install and remove Tk X error handlers
 * in order to catch X errors and deal with them.
 */
class ErrorHandler {
...
public:
    // constructor
    ErrorHandler(Display* display, int verbose = 1);

    // destructor
    ~ErrorHandler();

    // static version of error handler, called from Tk
    static int errorProc(ClientData clientData, XErrorEvent *errEventPtr);

    // install/remove an X error handler
    int install();
    int remove();

    // return 1 if errors occurred
    int errors();

    // reset the error flag
    int reset();
};
```

DESCRIPTION

This class is used to install and remove Tk X error handlers in order to catch X errors and deal with them.

Note that checking for X errors requires calling XSync(3X) to flush the X event queue. This could have performance and other implications, but is necessary due to the nature of the X window system client server communication (error messages could be delayed otherwise).

To use this class, you only need to declare a (local) variable of type ErrorHandler before doing something that might cause X errors. When you want to check if any X errors occured, you can call the "errors()" method. It returns true if any X errors occured:

```
    {
        ErrorHandler errorHandler(display, verbose_flag);
        // ... (something that might produce X errors)
        if (errorHandler.errors()) {
                // errors occurred ...
        }
    }
```

The constructor installs the error handler and the destructor removes it - the destructor for local variables is called automatically at the

```
end of the block or function where it is declared, and then the error
handler is automatically removed.

Since installing the error handler causes all X errors to be ignored,
it probably only makes sense to install one before certain operations
and remove it immediately afterwards (unless you are prepared to
handle X errors at any point in the application...).
```

METHODS

```
static int errorProc(ClientData clientData, XErrorEvent *errEventPtr)
        Static version of error handler, called from Tk.

int install();
int remove();
        Install or remove the X error handler.

int errors()
        Return 1 if errors occurred. A call to XSync() is made here to
        avoid delays due to X buffering.

int reset();
        Reset the error flag so that the "errors()" method will return
        0.
```

SEE ALSO

```
        CrtErrHdlr
```

```
- - - - - -
Last change: 07 May 99
```

4.2.3 HTTP(3)

NAME

HTTP - utility class for communicating with the HTTP

SYNOPSIS

```
#include "HTTP.h"

class HTTP {
...
public:
    HTTP();
    ~HTTP();
    int status();

    int get(const char* url);
    char* get(const char* url, int& nlines, int freeFlag = 1);
    int get(const char* url, ostream&);

    int post(const char* url, const char* data);
    int post(const char* url, const char* data, ostream&);

    int copy(ostream& os);
    int readline(char* ptr, int maxlen);
    char* getNext();
    const char* result();

    void feedback(FILE* f);
    FILE* feedback();

    const char* hostname();
    int fd();

    char* content_type();
    char* content_encoding() const;
    int content_length();

    const char* www_auth_realm() const;

    int authorizationRequired();

    static void authorize(const char* user, const char* passwd,
           const char* realm = NULL, const char* server = NULL);

    int html_error(istream& is);
    int html_error(char* s);

    static int allowUrlExec();
    static void allowUrlExec(int i);

    static const char* userAgent();
    static void userAgent(const char* s);

    static const char* authFile();
    static void authFile(const char* s);
};
```

DESCRIPTION

This class is used to access data over the network via HTTP. Methods

are available to get the data, given the URL and to step through the data line by line or return the entire result in a buffer. HTTP GET and POST operations are supported and well as HTTP redirects and proxy servers. Username/password authorization is also supported for restricted servers.

The syntax of a URL may be one of the following:

```
http://host:port/path        (:port is optional)

file://path

command args
```

In the first case, an HTTP GET (or POST) is done to retrieve the data for the URL. If the URL starts with "file:", the contents of the local file is returned. If the URL does not start with "file:" or "http:", it may be a local command to execute. In this case, the command is executed and the output of the command is returned. This last case is only allowed if explicitly enabled with the allowUrlExec() method. This should only be done if you know that it is safe to execute the command.

PROXY SERVERS

A proxy server, if available, is defined by the "http_proxy" environment variable. This should have the format:

```
http_proxy = "http://wwwcache.some.domain:port/"
```

A list of domains that do not require to be proxied (i.e. local machines etc.) is defined by the variable "http_noproxy". This should be a list of comma separated names, i.e.:

```
http_noproxy = "local.domain,national.domain"
```

If the given host is in one of these domains no proxy will be set up otherwise the member variables "proxyname_" and "proxyport_" will be defined (specifically proxyport_ will be set to -1 for no proxy).

HTTP REDIRECTS

This class supports HTTP redirects. If the result of a HTTP GET contains a header line matching: "Location: <URL>", then a recursive HTTP GET is done on the new URL and the result is returned to the caller in the normal way.

AUTHORIZATION

If an HTTP server returns the error "401 Authorization Required", the "realm" string from the message is noted and a special error message is generated:

```
"Authorization Required for <realm> at <hostname>"
```

Applications can look for this error message after an HTTP GET or check with the method: authorizationRequired() or www_auth_realm(). Typically, at this point, a user interface will ask for a username and passwd for the given realm and hostname. These can then be supplied by calling HTTP::authorize(username, passwd, realm, server). The last two arguments are optional, but if supplied, the information is saved in an authorization file (auth_file) for later reference. The username and password are encoded in the file, which has the following format

```
for each line:

    <server>:<realm>:<base64 encoded username:passwd>

The file is consulted in future sessions, so that the user doesn't
have to retype the information every time. The default location of the
file is ~/.http_auth and it is created with permissions so that
only the owner can read it.
```

METHODS

```
get(url)
        Do an HTTP GET with the given URL and position the read fd
        after the result HTTP header. Returns 0 if OK. Use readNext()
        to read the result lines.

get(url, nlines&, freeFlag)
        Do an HTTP get on the given URL and return the result as a
        buffer.  nlines is set to the number of result lines (use
        getNext() to fetch the result lines from the buffer.

get(url, ostream&)
        Do an HTTP get on the given URL and write the results to the
        given stream.

copy(ostream&)
        Copy the results of a previous get(url) to the given stream.

readline(ptr, maxlen)
        Read a line of the results after a call to get(url, nlines).

getNext()
        Return the next result line after a call to get(url) or NULL
        if there are no more lines. A null char is inserted in place of
        the newline so that the line can be treated as a single string.

result()
        Return a pointer to the result buffer from http.

feedback(file)
feedback()
        Set (or get) the file ptr to use for feedback during http
        transfers.  This can be used to give "netscape" like feedback
        in a progress bar while the transfer is in progress.

status()
        Return status after constructor.

hostname()
        Return the hostname of the http server for the last URL.

fd()
        Return file desc from last call to get(url).


content_type()
content_encoding()
content_length()
        Return http header values from last GET or POST.

www_auth_realm()
        If we see: WWW-Authenticate: Basic realm="somedomain" in the
        HTTP header, we assume a username and password are required
```

```
            and www_auth_realm_ is set to the "somedomain" value.

        authorizationRequired()
                Returns true if a username/passwd is needed for the previous
                HTTP GET operation.

        authorize(const char* user, const char* passwd,
                  const char* realm = NULL, const char* server = NULL);

                Set the username and password to use for authorization and, if
                realm and server are given, save an entry in the auth_file for
                later reference.

                This is normally called after an HTTP GET returned "401
                Authorization Required" and the user interface asked the user
                to type in a username and password, but could also be called
                at other times, if the necessary information is available.

                The server name can be obtained via the hostname() method
                after calling get(). The realm string is returned by the
                method www_auth_realm() after a get(). These values are also
                included in the error message generated when the "401
                Authorization Required" HTTP result is received.

        html_error(istream& is);
                Read an error message in HTML format and pass it on to
                error(), stripped of HTML syntax.

        html_error(char* s);
                Take an error message string in HTML format and pass it on to
                error(), stripped of HTML syntax.

        allowUrlExec()
        allowUrlExec(int)
                Get or set the flag value to allow a URL to be a local command.
                If the argument is non-zero, URLs not starting with "http:" or
                "file:" may be local commands to be executed.

        userAgent()
        userAgent(const char* s);
                Get or set the value of the HTTP User-Agent to use with
                requests (default: "SkyCat/1.0").

        authFile()
        authFile(const char* s)
                Get or set the value of the file used to remember
                authorization info (default: ~/.http_auth).
```

SEE ALSO

```
        error
```

```
- - - - - -
Last change: 07 May 99
```

4.2.4 Mem(3)

NAME

Mem - C++ class for managing shared, unshared or mmaped memory areas

SYNOPSIS

```
#include "Mem.h"

class Mem {
...
public:
    Mem();
    Mem(int size, int shmId, int owner, int verbose);
    Mem(int size, int useShm, int verbose = 0);
    Mem(const char *filename, int verbose = 0);
    Mem(const Mem& m);

    ~Mem();

    Mem& operator=(const Mem&);

    int shared() const;
    int shared(int share);

    static void cleanup();

    int length() const;
    int size() const;
    void* ptr() const;
    int shmId() const;
    int status() const;
    int verbose() const;
    void offset(long newOffset);
    long offset() const;
};

void Mem_cleanup(int);
```

DESCRIPTION

This class uses reference counting to help manage memory areas. The memory may be allocated with the C++ "new" operator, SysV shared memory or mmap. The class keeps track of who owns the memory, who is responsible for deleting it when no longer needed and how many references there are to the memory area. When there are no more references, we can safely detach a shared memory area and if we are the "owner", delete it. Constructor arguments and flags determine whether shared memory, mmap or normal (unshared) memory is allocated or whether existing shared memory is used. The state can be changed at any time, i.e.: shared memory can be changed to unshared and unshared to shared.

METHODS

Mem()

Default constructor, create empty, unshared memory.

Mem(size, shmId, owner, verbose)

Constructor: attach (if needed) to existing shared memory area. The size gives the expected size in bytes (for error checking) and shmId is the shared memory Id to use. If owner

```
        is non-zero, the memory will also be deleted when there are no
        more references.  If "verbose" is non-zero, debugging messages
        are printed on stdout.

Mem(size, useShm, verbose)
        Constructor: create a new memory area of the given size (in
        bytes), using shared shared memory if "useShm" is true.  If
        "verbose" is non-zero, debugging messages are printed on
        stdout.

Mem(filename, verbose)
        Constructor: uses mmap to map a file to a read-only mmap area.

Mem(mem)
        Copy constructor: only copies internal pointer.

~Mem()
        Destructor: detaches and/or deletes the memory, if needed,
        depending on the value of "owner" specified in the
        constructor.

Mem& operator=(mem)
        Assignment operator.

shared()
        Return true if the memory is shared.

shared(share)
        Force the memory to be shared (1) or not shared (0).

cleanup()
        Remove all "owned" shared memory areas (should be called
        before exit).

void offset(long newOffset)
        Set an offset in the memory area. two Mem objects may share
        the same memory area, both with different offsets. this is
        common when using mmap.

long offset() const
        Return the offset.

int length() const
        Return the working length of the memory (size minus offset).

size()
        Return the size of the memory area in bytes.

ptr()
        Return a pointer to the memory area.

shmId()
        Return the shared memory Id or -1 if the memory is not shared.

status()
        Return the status after the constructor has completed (as
        usual, 0 is OK, 1 means errors occured).

verbose()
        Return the value of the verbose flag.
```

SEE ALSO

```
      FitsIO, ImageData(3C++), ImageIO(3C++)


- - - - - -
Last change: 07 May 99
```

4.2.5 TclCommand(3)

NAME

TclCommand - C++ base class for implementing new Tcl commands

SYNOPSIS

```
#include <TclCommand.h>


class TclCommand {
protected:
    // tcl interpreter
    Tcl_Interp* interp_;

    // stream to evaluate tcl commands
    TclInterpStream tcl_;

    // status after constructor
    int status_;

    // class of tcl command (same as prefix for instname_)
    const char* cmdname_;

    // name of tcl command created for this object
    char* instname_;

    // used to generate unique tcl command name if name is specified as '#auto'
    static int seq_;

protected:
    // tcl command proc, called by tcl, calls teh correct member function
    static tclCmdProc(ClientData, Tcl_Interp* interp, int argc, char* argv[]);

    // tcl delete proc, called when tcl object is deleted
    static void tclDeleteProc(ClientData);

    // check the arg count for a subcommand
    int check_args(const char* name, int argc, int min_args, int max_args);

    // call a member function by name
    virtual int call(const char* name, int argc, char* argv[]);

    // append an integer result in tcl
    int append_result(int);

    // append a string result in tcl
    int append_result(char*);

    // return an integer value in tcl
    int set_result(int);

    // return a string value in tcl
    int set_result(char*);

    // report an error in tcl
    int error(char* msg);

public:

    // constructor
    TclCommand(Tcl_Interp*, const char* cmdname, const char* instname);
```

```
    // destructor
    virtual ~TclCommand();

    // return status of constructor
    int status() const { return status_; }

    // tcl delete sub command (always the same)
    int delete_(int argc, char* argv[]);

    TclInterpStream& tcl() {return tcl_;}

};
```

DESCRIPTION

TclCommand is a base class for C++ classes defining Tcl commands. It offers support for an [incr Tk] like interface. That is, Tcl commands defined in this framework can be declared as Tcl objects, where each object may have any number of methods (or subcommands). For example, you can define a Tcl command Dir in C++ and use it like this:

```
#create a Dir object called d
Dir d
foreach file [d lsfiles] {...}
foreach dir [d lsdirs] {...}

# The '#auto' syntax is also supported (generate a unique object name)
set d [Dir #auto]

# From within itcl classes, I usually use something like this:
Dir $this.d
$this.d lsfiles ...
```

The TclCommand class and classes derived from it each define the virtual member function

```
int call(const char* name, int argc, char* argv[])
```

to call a the member function that implements the named Tcl subcommand. The search starts at the bottom of the C++ inheritance hierarchy and works its way up to the base class. Each derived class checks a local static table of member functions (if necessary) to see if it implements the named method and if so, calls the method with the given arguments. If the given name does not correspond to a local method, control is passed to the next higher class in the inheritance hierarchy.

The member functions that implement Tcl subcommands all have the same signature. For example, the member function for the delete subcommand is defined in the base class for all Tcl objects like this:

```
// tcl delete sub command (always the same)
int delete_(int argc, char* argv[]);
```

EXAMPLE

An example of a simple Tcl command implemented in C++ is the Dir command, which has a number of subcommands for listing file and/or directories. The C++ class declaration for the Dir command looks like this:

```
#include "TclCommand.h"

/* This class declares the methods used to implement the Tcl "Dir"
 * command.  This command should be some what faster than "glob" or
 * "ls" for getting a list of "only files" or "only dirs" in a
 * directory.
 */

class TclDirCmd : public TclCommand {
private:
    DIR* opendir_(char* dirname);
    int isdir_(char* path);
    int isfile_(char* path);
public:

    // -- tcl subcommands --

    int ls_(int argc, char* argv[]);
    int lsfiles_(int argc, char* argv[]);
    int lsdirs_(int argc, char* argv[]);
    int cd_(int argc, char* argv[]);
    int pwd_(int argc, char* argv[]);

    // constructor
    TclDirCmd(Tcl_Interp*, const char* cmdname, const char* instname);
};

The C++ body file then looks something like this:
/*
 * declare a table of tcl subcommands
 * format: name, min_args, max_args, method
 */
static class TclDirSubCmds {
public:
    char* name; // method name
    int (TclDirCmd::*fptr)(int argc, char* argv[]);
    int min_args;    // minimum number of args
    int max_args;    // maximum number of args
} subcmds_[] = {
    {"ls", &TclDirCmd::ls_,  1,  1},
    {"lsfiles", &TclDirCmd::lsfiles_,  1,  1},
    {"lsdirs",  &TclDirCmd::lsdirs_,   1,  1},
    {"cd", &TclDirCmd::cd_,  1,  1},
    {"pwd",&TclDirCmd::pwd_, 0,  0},
};


/*
 * Call the given method in this class with the given arguments
 */
int TclDirCmd::call(const char* name, int argc, char* argv[])
{
    for(int i = 0; i < sizeof(subcmds_)/sizeof(*subcmds_); i++) {
        TclDirSubCmds* t = &subcmds_[i];
        if (strcmp(t->name, name) == 0) {
            if (check_args(name, argc, t->min_args, t->max_args) != TCL_OK)
                return TCL_ERROR;
            return (this->*t->fptr)(argc, argv);
        }
    }
    return TclCommand::call(name, argc, argv);
}
```

SEE ALSO

TkWidget, TkImage(3C++), TclInterpStream(3C++)

- - - - - -

Last change: 07 May 99

4.2.6 TkImage(3)

NAME

TkImage - A C++ base class for implementing Tk image types

PARENT CLASS

TclCommand

DERIVED CLASSES

RtdImage

SYNOPSIS

```
#include "TkImage.h"

/*
 * This class or a class derived from it is used to hold the values
 * modified by the configure image command
 */
class TkImageOptions {
public:
};


/*
 * This is the base class for classes defining tk images from
 * C++ classes.
 * In this class, only one instance of an image may be displayed,
 * in a window, however multiple views of the image with varying
 * sizes may be displayed (I didn't see any need for exact copies
 * of the same image in the same size...)
 */
class TkImage : public TclCommand {
protected:
    Tk_ImageMaster master_;     // passed from the Tk imaging routines

    TkImageOptions* optionsPtr_; // ptr to class or struct holding option values
    Tk_ConfigSpec* configSpecsPtr_; // ptr to configSpecs array for Tk options
    int refCount_;              // Number of instances that share this
                                // image (may only be one here).

    Tk_Window tkwin_;           // Window in which the image will be displayed.
    Display *display_;          // X token for the window's display
    Visual *visual_;            // X visual for window
    GC gc_;                     // Graphics context for copying to screen
    Pixmap pm_;                 // Pixmap for storing the image

    int width_;                 // total width of image
    int height_;                // total height of image
    int pixw_;                  // width of X pixmap, if used
    int pixh_;                  // height of X pixmap, if used
    int depth_;                 // depth of screen

   char* pclass_;              // if specified, restrict images to only be displayed
                                // in widgets of that class

    int update_pending_;        // flag: true if image needs to be updated

    int initialized_;           // flag: true after image has been initialized
                                // (configured, after constructor is done)
```

```
    // -- member functions --

    // do first time image configuration
    virtual int initImage(int argc, char* argv[]);

    // configure the image with cmd line options.
    virtual int configureImage(int argc, char* argv[], int flags = 0);

    // these are called indirectly by the Tk imaging routines
    virtual TkImage* getImage(Tk_Window);
    virtual void displayImage(Drawable, int imageX, int imageY,
                              int width, int height,
                              int drawableX, int drawableY) = 0;

    // update the image display eventually
    virtual void imageChanged();

    // make the X graphics context
    void makeGC();

    // set the image size and create/update a pixmap, if use_pixmap is 1
    int setImageSize(int width, int height, int use_pixmap, int pixw, int pixh);

    // utility method: equivalent of Tk "update idletasks" command
    void updateIdleTasks();

    // call a member function by name
    virtual int call(const char* name, int len, int argc, char* argv[]);

public:

    // constructor
    TkImage(Tcl_Interp* interp, char* cmdname, char* instname,
                Tk_ConfigSpec* specs, TkImageOptions& options,
                Tk_ImageMaster master, char* pclass = NULL);

    // destructor
    virtual ~TkImage();

    // the following static methods are called by the Tk image handling routines
    static ClientData GetImage(Tk_Window, ClientData);
    static void DisplayImage(ClientData, Display*, Drawable,
                             int imageX, int imageY,
                             int width, int height,
                             int drawableX, int drawableY);
    static void FreeImage(ClientData, Display*);
    static void DeleteImage(ClientData);

    // implement the configure Tk command
    virtual int configureCmd(int argc, char* argv[]);

    // called for the cget image command
    virtual int cgetCmd(int argc, char* argv[]);

};
```

DESCRIPTION

TkImage is a base class for a class defining a new TkImage type. The image type defined here differs slightly from standard Tk image types, such as photo or bitmap in that it is only allowed to display a given image in a single widget and you can optionally specify what type of

widget that should be.

The intent here is that you specify that the image can only be in, say, a canvas window. Since a given image "instance" can only be displayed in one canvas window (as far as Tk is concerned), you can use the scrolling offsets of the canvas to implement intelligent scrolling. It is still possible to display an image in multiple widgets, but this must be handled in the derived class (see RtdImage for an example). The restriction comes about, because of the way Tk handles images in multiple widgets. Normally, if one changes size, the others do as well. This is not necessarily what you always want in an image processing application. You may want to have the same image at different sizes sharing the same data.

This class does some of the abstract work for dealing with Tk images. It declares some of the static member functions that Tk calls for image handling (GetImage, DisplayImage, FreeImage and DeleteImage) and calls virtual member functions where necessary to handle the actual display of the image.

This class also defines and implements the "configure" and "cget" image commands. The derived class passes the TkImage constructor information about the configuration options (optionsPtr_ and configSpecsPtr_) for use in these subcommands. The derived class may still need to redefine the "configureImage" method inorder to handle certain options, but most of the work is done in this base class.

INHERITANCE

Note that each class in this class hierarchy defines the virtual method "call" to call a member member function, given the name of the Tcl subcommand. The search starts in the derived class and if a method is not found there, its parent class is searched and so on. In this way, each level in the class hierarchy can define Tcl subcommands (see TclCommand).

MEMBER VARIABLES

This class keeps track of the following information for the derived class:

- Tk image record (master_)
- Configuration options (optionsPtr_, configSpecsPtr_)
- Widget Reference count (only allowed to be 1) (refCount_)
- Tk window where image is displayed (tkwin_)
- X Display for window (display_)
- X Visual for window (visual_)
- Graphics Context (gc_)
- X Pixmap for drawing (not always used) (pm_)
- Width and Height of image (width_, height_)
- Width and Height of Pixmap (pixw_, pixh_)
- Allowed widget type for image (pclass_)
- Update pending flag (to force image update) (update_pending_)
- Initialization flag (set after first create/config) (initialized_)

METHODS

This class also defines the following methods for use by the derived class:

virtual int initImage(int argc, char* argv[])
: Do first time image configuration. This procedure needs to be called from the derived class constructor to complete the image initialization. This can't be done in the constructor

```
                here since the options_ struct wouldn't be initialized yet.

        virtual void imageChanged()
                This method should be called when the image has changed and
                should be redrawn eventually.

        int setImageSize(int width, int height, int use_pixmap, int pixw, int pixh)
                Set the image dimensions to the given width and height and if
                use_pixmap is 1, create or update a pixmap to have the same
                dimensions.

        void updateIdleTasks()
                Utility method: equivalent of Tk "update idletasks" command.
                Process all pending display events.
```

SEE ALSO

TclCommand, RtdImage(3C++), image

- - - - - -

Last change: 07 May 99

4.2.7 TkWidget(3)

NAME

TkWidget - C++ base class for implementing new Tk Widgets

PARENT CLASS

TclCommand

DERIVED CLASSES

TkTree

SYNOPSIS

```
#include "TkWidget.h"


/*
 * This is the base class for classes defining tk widgets from
 * C++ classes
 */
class TkWidget : public TclCommand {
protected:
    Tk_Window tkwin_;   // widget's Tk window
    Display *display_;  // X token for the window's display

    char* pname_;       // name of parent window
    char* wclass_;      // this widget's class

    Tk_ConfigSpec* configSpecsPtr_; // ptr to widget's config struct
    TkWidgetOptions* optionsPtr_;   // ptr to struct holding option values

    int redraw_pending_;   // flag: true if widget needs to be redrawn


    // do first time widget configuration
    virtual int initWidget(int argc, char* argv[]);

    // configure the widget with cmd line options.
    virtual int configureWidget(int argc, char* argv[], int flags = 0);

    // If not already scheduled, schedule the widget to be redrawn
    virtual int eventually_redraw();

    // this method may be redefined in a derived class to redraw
    // the widget.
    virtual void redraw();

    // called for DestroyNotify events in the widget
    virtual void destroyNotify(XEvent *eventPtr);

    // called for ConfigureNotify events (resize)
    virtual void configureNotify(XEvent *eventPtr);

public:

    // constructor: pclass is the expected parent window class type,
    // the specs and options args are used to process the command line
    // args and for the configure widget command.
    TkWidget(Tcl_Interp*, const char* pclass,
             Tk_ConfigSpec* specs, TkWidgetOptions& options,
```

```
                int argc, char* argv[]);

        // destructor
        virtual ~TkWidget();

        // call a member function by name
        virtual int call(const char* name, int len, int argc, char* argv[]);

        virtual int configureCmd(int argc, char* argv[]);

        // called for the cget widget command
        virtual int cgetCmd(int argc, char* argv[]);

        // Redraw the widget by calling the virtual redraw_ method
        // which may be defined in a derived class.
        static void redrawWidget(ClientData);

        // called for StructureNotify events in the widget
        static void structureNotify(ClientData clientData, XEvent *eventPtr);

        // invoked by Tk_EventuallyFree to clean up widget
        static void destroyProc(ClientData clientData);
    };
```

DESCRIPTION

TkWidget is a base class for C++ classes defining Tk widgets. In addition to the features inherited from its base class TclCommand, TkWidget adds support for creating and configuring Tk widgets.

The configure (or config) and cget subcommands are implemented here automatically, so that derived classes don't have to handle this. The derived classes only need to define two structures required by Tk for widgets:

One is the Tk_ConfigSpec structure, a static struct describing the widget options.

The other is a non-static member struct for holding the values for widget configuration options. This struct must be derived from TkWidgetOptions, which is just a dummy struct used for passing the actual struct as a parameter.

SEE ALSO

TclCommand, TkTree(3C++)

- - - - - -

Last change: 07 May 99

4.2.8 util(3)

NAME

util - Collection of C++ utility routines

SYNOPSIS

```
#include "util.h"

char** copyArray(int len, char** ar);
char* stripWhiteSpace(char* p);
const char* fileSuffix(const char* p);
const char* fileBasename(const char* p);
const char* fileRealname(const char* filename, char* buf, int buflen);
int fileSize(const char* filename);
int fileAbsPath(const char* filename, char* path, int pathlen, int& flag);
int readUnbufferedLine(int fd, char* ptr, int maxlen);
int readUnbufferedBytes(int fd, char* ptr, int nbytes);
int writeUnbufferedLine(int fd, char* ptr);
int writeUnbufferedBytes(int fd, char* ptr, int nbytes);
int localSockConnect(int& sock, int port);
int localSockListen(int& sock, int& port);
```

DESCRIPTION

A number of general purpose C++ routines have been gathered here. Most of these should probably be in more specialized C++ String or File classes.

ROUTINES

```
copyArray(len, ar);
        Make a copy of the given string array in a single buffer.

stripWhiteSpace(p);
        Strip white space from string p by returning offset in string
        and setting trailing white space to '\0'.

fileSuffix(p);
        Return the suffix of the string p (part following ".", if
        any).

fileBasename(p);
        Return the basename of the filename p (part following last
        "/", if any).

fileRealname(filename, buf, buflen);
        Get the real name of the given file, which may be the name of
        a symbolic link. If the file exists, the resolved name is
        written to the given buffer and returned, otherwise a pointer
        to the original filename is returned. No error message is
        generated here.

fileSize(filename);
        Return the size of the file in bytes or -1 on error.

fileAbsPath(filename, path, pathlen, flag);
        If filename is not an absolute path (starting with '/'), write
        an absolute path for it to "path". Sets flag to 1 if path was
        changed.

readUnbufferedLine(fd, ptr, maxlen);
        Read a line from the given file descriptor one byte at a time,
```

```
            with no buffering, into the given buffer of the given length
            (for sockets and pipes, stdin, ...).

        readUnbufferedBytes(fd, ptr, nbytes);
            Read nbytes bytes from the given file descriptor into the
            given buffer, with no buffering.

        int writeUnbufferedLine(int fd, char* ptr);
            Write the given buffer (ptr) using unbuffered I/O to the given
            file descriptor, followed by a newline.

        int writeUnbufferedBytes(int fd, char* ptr, int nbytes);
            Write nbytes bytes to the given file descriptor using
            unbuffered I/O (for sockets and pipes, stdin,...).

        int localSockConnect(int& sock, int port);
            Open a socket connection on the given port on teh local host.

        int localSockListen(int& sock, int& port);
            Open a socket and start listening on the given port (or choose
            port, if 0).



- - - - - -
Last change: 07 May 99
```

4.2.9 ITCL CLASSES

4.2.10 Batch(n)

NAME

Batch - Class to evaluate Tcl commands in a separate process.

NAMESPACE

util

PARENT CLASS

util::FrameWidget

SYNOPSIS

Batch <path> ?options?

DESCRIPTION

This class is used to evaluate a set of Tcl commands in a separate process, so that it can be interrupted. No exec is done, so the process is just a copy of the current one and has a copy of the memory and open files. .

WIDGET OPTIONS

-command
: Command to evaluate when process is done. Called with 2 args: status and result (or error message).

-debug
: Flag: if true run commands in the foreground for better debugging.

-tmpfile
: Temp file for results.

PUBLIC METHODS

bg_eval {cmd}
: Evaluate the given tcl commands in the background, so that they can be interrupted. The option $command is evaluated when the results when done.

fg_eval {cmd}
: Evaluate the given command in the foreground (useful for debugging) The option $command is evaluated when the results when done.

interrupt {}
: Interrupt the current search .

PROTECTED METHODS

read_pipe {rfd wfd}
: Read the pipe from the child process.

PROTECTED VARIABLES

bg_pid_
: Process id of background process.

COMMON CLASS VARIABLES

count_

Count used for filename generation.

SEE ALSO

FrameWidget(n)

- - - - - -

Last change: 07 May 99

4.2.11 ButtonFrame(n)

NAME

ButtonFrame - Widget displaying a frame with some standard buttons.

NAMESPACE

util

PARENT CLASS

util::FrameWidget

SYNOPSIS

ButtonFrame <path> ?options?

DESCRIPTION

This class is an itcl widget for displaying a frame with the standard buttons like "OK Cancel", with options for configuring other labels and actions.

WIDGET OPTIONS

-ok_label

Labels and commands for the standard buttons.

PUBLIC METHODS

append {label cmd}

Add a button.

buttonconfig {label args}

Add a button to the row or configure an existing button.

PROTECTED VARIABLES

but_

Array mapping label name to button name.

seq_

Counter for button names.

SEE ALSO

FrameWidget(n)

- - - - - -

Last change: 07 May 99

4.2.12 CanvasDraw(n)

NAME

CanvasDraw - Add interactive drawing capabilities to a Tk canvas.

NAMESPACE

util

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

CanvasDraw <path> ?options?

DESCRIPTION

CanvasDraw is an [incrTcl] widget class that an application can use to add interactive drawing capabilities to any Tk canvas. Since it is a subclass of TopLevelWidget, it creates its own toplevel window containing a drawing mode selection section and an option section for setting colors, fonts, line thickness, arrow types, etc. The window can be hidden on startup by specifying the "-withdraw 1" option inherited from TopLevelWidget.

ITK COMPONENTS

bot
: Button frame at bottom of window.

clear
: Clear button.

close
: Close button.

delete
: Delete button.

drawmodes
: Frame containing drawing mode buttons.

options
: Frame containing option menubuttons for graphic items.

$drawing_mode
: Drawing mode buttons (anyselect, region, line, rectangle, oval, polyline, polygon, text).

$menu
: LabelMenu(n) items for changing options on graphic objects. (Width Arrow ArrowShape Fill Outline Stipple Font Smooth).

WIDGET OPTIONS

-arrowshape
: Default arrow shape for lines.

-arrowtype

```
       Default arrow type for lines.

-bbox
       Specify bounding box of interactive drawing area as a list {x0 y0
       x1 y1}.

-canvas
       Canvas window.

-clipping
       Flag: if true, keep graphics within the specified bounding box
       (-bbox option).

-colors
       List of colors for menu.

-defaultcursor
       Default cursor when not drawing.

-drawcursor
       Cursor when drawing graphics.

-fillcolor
       Default fill color for drawing.

-fonts
       Default list of fonts for font menu.

-gripfill
       Fill color of selection grips.

-gripoutline
       Outline color of selection grips.

-gripwidth
       Size of selection grips.

-linecursor
       Cursor displayed over simple lines for resizing.

-linestipple
       Default stipple for lines.

-linewidth
       Default line width for drawing.

-outlinecolor
       Default color for outlines.

-regioncommand
       Tcl command to evaluate whenever a "region" object is created, if
       no specific "create" command was specified.

-regioncursor
       Cursor when creating region items (dashed box marking a region).

-show_object_menu
       Flag: if true, display menus over graphic objects when selected
       with <3>.

-show_selection_grips
       Flag: if true, display selection grips when an item is selected.
```

```
-smooth
       Default value for smooth option.

-stipplepattern
       Default stipple pattern for drawing.

-textcursor
       Cursor when creating text items.

-textfont
       Default text item font.

-withrotate
       If true (default) use polygons for rectangles to support rotation
       in Tcl (Tk doesn't support rotating, but polygons can be rotated
       in Tcl code).  Subclasses that define rotation by adding new
       canvas objects can set this option to 0.

-withtoolbox
       If true (default) create the GUI interface (toolbox), otherwise
       don't.
```

PUBLIC METHODS

```
add_menuitems {m}
       Create the menu items in the given menu.

add_notify_cmd {id cmd {update_flag 0}}
       Arrange to have cmd evaluated whenever the given object is moved
       or resized. id is the canvas id of the object, If update_flag is
       1, $cmd is also evaluated while the item is being moved or
       resized, otherwise only when the operation is done.

add_object_bindings {id {target_id ""}}
       Set up the canvas object bindings. target_id defaults to id.  If
       target_id is specified, it will receive the events instead of id.

adjust_line_selection {id}
       Adjust the selection handles for the given simple line.

adjust_object_selection {id}
       Adjust the selection handles for the given object to fit the new
       size/pos.

check_deselect {}
       If the mouse is not over an object, deselect all objects.

clear {}
       Delete all canvas items.

clip {px py}
       Clip the given point to keep it over the drawing area.

config_selected {option value}
       Apply the given option to the selected canvas items.

create_done {x y}
       Finish creation of the current object and call the "create"
       command, if one was specified, otherwise, for region items, call
       the region command, if defined.

create_object {x y}
       Create a new object in the canvas.
```

```
delete_object {id}
       Delete the given canvas item.

delete_selected_objects {}
       Delete the selected canvas items.

deselect_object {id}
       Deselect the given object.

deselect_objects {}
       Deselect all canvas objects.

flipx {item maxx}
       Flip the named item(s) on the X axis by subtracting the x
       coordinates from $maxx.

flipy {item maxy}
       Flip the named item(s) on the Y axis by subtracting the y
       coordinates from $maxy.

item_has_tag {item tag}
       Return 1 if the given item has the given tag.

item_is_selected {item}
       Return 1 if the given item is currently selected.

remove_notify_cmd {id}
       Remove the notify command for the given canvas id.

reset_cursor {}
       Set the cursor for the canvas window back to the previous one,
       unless we are in the middle of a resize operation.

rotate {item}
       Exchange the X and Y coords for the given item (not really
       rotate...).

rotate_point {x y cx cy angle}
       Rotate the given point about the given center point by the given
       angle (in radians) and return a list {x y} as the result.

select_object {id {any 0}}
       Select the given object by drawing 8 little grips on it, or for
       text objects, if any==1 and the selection type is "anyselect",
       select the text in the object.  The grips have the tag "grip" and
       the object gets the tag "$w_.selected".

select_only_object {id x y}
       Select only the given object.

select_region {x0 y0 x1 y1}
       This method is called when a region object has been created to
       select a region of the canvas.  All objects in the region are
       selected and if a region command option was specified, the command
       is evaluated with the bounding box of the region.

selected_items {}
       Return a list of all of the currently selected items.

set_arrowshape {list}
       Set the arrow shape default and for any selected objects.
```

```
set_arrowtype {t}
       Set the arrow type default and for any selected objects.

set_cursor {cursor}
       Set the cursor for the canvas window and save the previous cursor
       for later restoration.

set_drawing_mode {type {create_cmd ""}}
       Set the type of object to draw (line, oval, ...) If create_cmd is
       specified, it is called with the coordinates of the new object
       when its creation is completed.

set_fillcolor {c}
       Set the fill color default and for any selected objects note: if
       the stipple is invisible (see through), make it solid to make sure
       fill is visible.

set_linewidth {w}
       Set the line width default and for any selected objects.

set_menu_state {state}
       Enable or disable the drawing items in the menu (enable is state
       is "normal", disable if "disabled".

set_outlinecolor {c}
       Set the outline color default and for any selected objects.

set_smooth {bool}
       Set the smooth option for drawing .

set_stipplepattern {index}
       Set the stipple pattern default and for any selected objects index
       is the pattern number (corresponding to pat$index.xbm).

set_textfont {f}
       Set the text font  default and for any selected objects.
```

PROTECTED METHODS

```
add_poly_point {x y}
       Add a point to the curent polygon/polyline.

check_done {x y {update_method "update_object"}}
       Check if object creation is done: if the button has moved since it
       was pressed, we are done, otherwise dragging continues without the
       button pressed.

       update_method is the method to call to update the object, if
       needed.

create_arc {x y}
       Create  and return a new arc object.

create_freehand {x y}
       Create  and return a new freehand object.

create_label {x y}
       Create and return a new label (entry) object.

create_line {x y}
       Create  and return a new line object.

create_oval {x y}
```

```
       Create  and return a new oval object.

create_polygon {x y}
       Create  and return a new polygon object.

create_polyline {x y}
       Create  and return a new polyline object.

create_rectangle {x y}
       Create and return a new rectangle object (actually create a
       polygon object in the shape of a rectangle, since it is more
       flexible and can be rotated).

create_region {x y}
       Create and return a new region (dashed rectangle marking a region
       of the canvas).

create_text {x y}
       Create  and return a new text object.

create_text_done {x y}
       Finish creation of the current text object.

draw_line_selection_grips {id}
       Draw the selection grips for a simple line.  Lines get one grip at
       each end.

draw_selection_grips {id {type ""}}
       Draw the object selection grips (small boxes similar to those used
       by FrameMaker and used to move and resize the object).  Lines (see
       above) get one at each end, other objects (here) get one at each
       corner and side of the object's bounding box. If type is "polygon"
       or "line", rotation bindings are also enabled (with
       <Control-...>).

end_move {id}
       This method is called when a move operation is done.

end_resize {id}
       Stop resizing the selected object.

end_resize_line {id}
       Stop resizing the selected line.

end_rotate {id}
       Stop rotating the selected object.

get_line_color {}
       Return a color for a line that will be visible.

init {}
       This method is called after the constructor have completed.

make_drawmode_frame {}
       Create the frame for selecting draw types  (line, oval, ...) in
       the given frame.

make_object_menu {}
       Create the canvas item menu.

make_options_frame {}
       Create the draw options frame (line width, color, etc...).
```

```
mark {x y}
       Mark (remember) the current position.

move_object {id x y}
       Move the given object.

post_object_menu {x y}
       Post the object menu at the mouse cursor position.  If an item is
       selected, post the menu with the items enabled.

resize_label {text rect tag}
       Resize the rectangle for a label to fit the text.

resize_line {id x y side constrain}
       Resize the given line.

start_resize {id x y side}
       Start resizing the selected object (19.9.96: added changes from
       pbiereic).

start_rotate {id x y}
       Start rotating the selected object.

toggle_select_object {id x y}
       If the object is selected, deselect it, otherwise select it.

update_display_from_object {id}
       Update the display to show the options for the given canvas item.

update_freehand {x y}
       Update the current  freehand object in the canvas with the new x,
       y end points.

update_line {x y {constrain 0}}
       Update the current line object in the canvas with the new x, y end
       points.

update_object {x y {constrain 0}}
       Update the current object in the canvas with the new x, y end
       points.

update_poly_object {x y}
       Update the current polygon/polyline object in the canvas with the
       new x, y end points.

update_rectangle {x y {constrain 0}}
       Update the current rectangle (or polygon, so we can rotate it)
       object in the canvas with the new x, y end points.

update_region {x y {constrain 0}}
       Update the current region object (dashed rectangle used for
       marking a region on the canvas).

update_resize {id x y side constrain}
       Update the resizing of the selected object (updated with changes
       from pbiereic, 19.9.96).

update_rotate {id x y}
       Update the rotating of the selected object.
```

PROTECTED VARIABLES

```
arrowshapes_
```

```
       Available arrow shapes for canvas lines (corresponds to bitmaps
       named ar$1_$2_$3).

canvasX_
       Strings used in bindings to translate %x to canvas coords.

canvasY_
       Strings used in bindings to translate %y to canvas coords.

canvas_
       Canvas widget.

create_cmd_
       Command to evaluate when the current item has been created.

dragx_
       Flag: true if resizing on X axis.

dragy_
       Flag: true if resizing on Y axis.

drawing_
       Flag, true if drawing currently.

drawing_mode_
       Current draw type.

drawing_modes_
       List of drawing types .

endx_
       Ending x coord of object being drawn.

endy_
       Ending y coord of object being drawn.

menu_
       Saved menu (optional, from caller's menubar) with drawing commands
       for setting the state of the menu items.

moved_
       Array(id) of flag: set if object with id was moved.

notify_
       Array(id) of tcl command to evaluate when object given by canvas
       id is moved or scaled.

notify_update_
       Same as notify_ above, but indicates command should be evaluated
       while item is being moved or resized.

obj_coords_
       List of coords for current object being created.

obj_id_
       Canvas id of the current canvas item.

object_menu_
       Popup menu for canvas items.

option_map_
       Array(canvasID,option) of option to use instead for that id.
```

```
posx2_
       X2 pos for resize operations.

posx_
       X pos for resize operations.

posy2_
       Y2 pos for resize operations.

posy_
       Y pos for resize operations.

prev_cursor_
       Saved previous cursor for canvas.

regions_
       Array(canvasId) of region ids, used to keep these from rotating.

resizing_
       Flag: true if currently resizing an item.

rotate_angle_
       Current rotation angle of object.

rotate_cursor_
       Cursor to use in rotate mode.

rotate_cx_
       Center X coordinate of object being rotated.

rotate_cy_
       Center Y coordinate of object being rotated.

startx_
       Starting x coord of object being drawn.

starty_
       Starting y coord of object being drawn.

stipple_index_
       Index of stipple bitmap (0 .. 15, 7 is see through, see
       -stipplepattern).

tag_count_
       Counter used for compound objects to generate canvas tags.

x0_
       X0 of bounding box of interactive drawing area.

x1_
       X1 of bounding box of interactive drawing area.

xoffset_
       X offset for move operations.

y0_
       Y0 of bounding box of interactive drawing area.

y1_
       Y1 of bounding box of interactive drawing area.

yoffset_
       Y offsets for move operations.
```

COMMON CLASS VARIABLES

cursors_
: Array(direction={n,w,e,s,nw,ne,sw,se}) of cursor names for selection handles.

pi_
: Const PI.

rad_
: Const PI/180.

SEE ALSO

TopLevelWidget(n)

- - - - - -

Last change: 07 May 99

4.2.13 CanvasPrint(n)

NAME

CanvasPrint - Popup dialog for printing the contents of a Tk canvas

NAMESPACE

util

PARENT CLASS

util::PrintDialog

SYNOPSIS

CanvasPrint <path> ?options?

DESCRIPTION

This class extends the PrintDialog class to be able to print the contents of a canvas as postscript. Note: this class does not support printing images embedded in the canvas.

WIDGET OPTIONS

-bot_left
Header text to appear at bottom left.

-bot_right
Header text to appear at bottom right.

-canvas
Canvas widget.

-fit_to_page
Flag, if true, scale output to fit on page.

-header_font
Header/footer fonts.

-pageheight
Page height, used when fit_to_page is 1.

-pagewidth
Page width, used when fit_to_page is 1.

-show_headers
Flag, if true, insert headers before printing.

-top_left
Header text to appear at top left.

-top_right
Header text to appear at top right.

-x0
X0 coordinate of area of canvas to print.

-x1
X1 coordinate of area of canvas to print.

```
-y0
       Y0 coordinate of area of canvas to print.

-y1
       Y1 coordinate of area of canvas to print.
```

PROTECTED METHODS

```
add_headers {}
       Add header and footer labels above/below draw area by temporarily
       inserting the text .

init {}
       This method is called after all options have been evaluated.

print {fd}
       Print the contents of the canvas to the open filedescriptor.

rm_headers {}
       Remove the headers, if any and restore the original state.

toggle_fit_pagesize {}
       Called when the "Fit on page" button is pressed.
```

PROTECTED VARIABLES

```
x0
       Saved -x0 option value.

x1
       Saved -x1 option value.

y0
       Saved -y0 option value.

y1
       Saved -y1 option value.
```

SEE ALSO

PrintDialog(n)

- - - - - -
Last change: 07 May 99

4.2.14 CanvasWidget(n)

NAME

CanvasWidget - Itcl class based on the Tk canvas widget

NAMESPACE

util

PARENT CLASS

util::FrameWidget

SYNOPSIS

CanvasWidget <path> ?options?

DESCRIPTION

This is an itcl class based on the Tk canvas widget. It combines a canvas window in a frame with optional scrollbars.

SEE ALSO

FrameWidget(n)

- - - - - -
Last change: 07 May 99

4.2.15 CheckEntry(n)

NAME

CheckEntry - Itcl widget combining a checkbutton and an entry

NAMESPACE

util

PARENT CLASS

util::FrameWidget

SYNOPSIS

CheckEntry <path> ?options?

DESCRIPTION

This class defines an Itk widget combining a checkbutton and an entry.

ITK COMPONENTS

check
: Tk checkbutton widget.

entry
: Tk entry widget.

STANDARD OPTIONS

-text

WIDGET OPTIONS

-command
: The command for <Return> in the entry.

-value
: Set the value in the entry.

-variable
: Optionally specify the trace variable to use.

PUBLIC METHODS

get {}
: Get the value in the entry.

PROTECTED METHODS

_command_proc {}
: called for Return in the entry.

PROTECTED VARIABLES

variable_
: Trace var for checkbutton.

SEE ALSO

```
       FrameWidget(n)



- - - - - -
Last change: 07 May 99
```

4.2.16 ChoiceDialog(n)

NAME

ChoiceDialog - Dialog to display a message and get choice from the user

NAMESPACE

util

PARENT CLASS

util::DialogWidget

SYNOPSIS

ChoiceDialog <path> ?options?

DESCRIPTION

A ChoiceDialog is a dialog widget used to display a message and get a choice from the user.

ITK COMPONENTS

choice

LabelChoice(n) Itk widget.

STANDARD OPTIONS

-choice -cols -rows -value

PUBLIC METHODS

set_choice {choice}

Called when a choice is made.

PROTECTED METHODS

set_result {}

This method is redefined here to change the value that is returned from activate to be the contents of the choice widget.

SEE ALSO

DialogWidget(n)

- - - - - -

Last change: 07 May 99

4.2.17 Chooser(n)

NAME

Chooser - A simple widget for selecting items based on filenames

NAMESPACE

util

PARENT CLASS

util::FrameWidget

SYNOPSIS

Chooser <path> ?options?

DESCRIPTION

Chooser is a simple itcl widget for selecting items based on files with a given suffix in a given directory.

ITK COMPONENTS

list

ListboxWidget(n) for displaying choice.

STANDARD OPTIONS

-background -font -foreground -height -hscroll -title -vscroll -width

WIDGET OPTIONS

-command

Tcl command to evaluate with selected file name.

-default

Default selection.

-dir

Directory for files.

-files

List of files (complete pathnames, usually output of [glob $dir/*.$suffix].

-suffix

Suffix for files.

PUBLIC METHODS

set_choice {file}

Change the selection to the given file.

PROTECTED METHODS

choose {}

Set new values.

PROTECTED VARIABLES

listbox_

Internal listbox widget.

SEE ALSO

FrameWidget(n)

- - - - - -

Last change: 07 May 99

4.2.18 DialogWidget(n)

NAME

DialogWidget - Base class of dialog widget classes

NAMESPACE

util

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

DialogWidget <path> ?options?

DESCRIPTION

A DialogWidget is an itk widget for creating dialog windows. A dialog window here is a window with a message at top and a row of buttons at bottom. Through inheritance, you can add widgets inbetween.

ITK COMPONENTS

bitmap
: Optional bitmap label.

bitmapf
: Optional bitmap frame.

bot
: Button frame at bottom.

def
: Default frame (for image, bitmap, message...).

default
: Frame surrounding the default dialog button.

ext
: Extension frame (for subclasses).

image
: Optional image label.

imagef
: Optional image frame.

text
: Optional message text.

top
: The top frame has one frame for the message and bitmap and another for extensions defined in derived classes.

button$i
: Components for buttons specified with the -buttons option. For example "buttonOK" for the OK button.

STANDARD OPTIONS

-background -justify -messagefont -messagewidth

WIDGET OPTIONS

-bitmap
: Optional bitmap to display to left of message.

-buttons
: One or more strings to display in buttons across the bottom of the dialog box.

-default
: Index of button that is to display the default ring (-1 means none).

-image
: Optional image to display above the message.

-modal
: Flag: if true, grab the screen.

-text
: Text of message (truncated if too long).

-title
: Title to display in dialog's decorative frame.

PUBLIC METHODS

activate {}
: Display the window and return the user's selection.

PROTECTED METHODS

add_buttons {}
: Create a row of buttons at the bottom of the dialog.

init {}
: This method is called after the options have been evaluated.

set_result {}
: This method may be redefined in a derived class to change the value that is returned from activate (default is the number of the button selected).

PROTECTED VARIABLES

variable_
: Trace variable name.

SEE ALSO

TopLevelWidget(n)

- - - - - -

Last change: 07 May 99

4.2.19 DoubleList(n)

NAME

DoubleList - Widget displaying two lists with arrows between them

NAMESPACE

util

PARENT CLASS

util::FrameWidget

SYNOPSIS

DoubleList <path> ?options?

DESCRIPTION

A DoubleList is an itcl widget for displaying two lists with arrows between them for moving items back and forth.

ITK COMPONENTS

left
: Left list component (see ListboxWidget(n)).

right
: Right list component (see ListboxWidget(n)).

STANDARD OPTIONS

-exportselection -height -hscroll -lefttitle -righttitle -selectmode -vscroll -width

WIDGET OPTIONS

-updown
: Flag: if true, also display up and down buttons for moving list items vertically.

PUBLIC METHODS

clear {}
: Make the lists empty.

move_down {}
: Move the selected elements from the right to the left list.

move_left {}
: Move the selected elements from the right to the left list.

move_right {}
: Move the selected elements from the left to the right list.

move_up {}
: Move the selected elements from the right to the left list.

SEE ALSO

FrameWidget(n)

```
- - - - - -
Last change: 07 May 99
```

4.2.20 DoubleTableList(n)

NAME

DoubleTableList - Widget displaying two TableLists with arrows between them.

NAMESPACE

util

PARENT CLASS

util::FrameWidget

SYNOPSIS

DoubleTableList <path> ?options?

DESCRIPTION

A DoubleTableList is an itcl widget for displaying two TableLists with arrows between them for moving items back and forth. You can access the left and right TableLists as itk components "left" and "right".

ITK COMPONENTS

left
: Left table component (TableList(n)).

right
: Right table (see TableList(n)).

STANDARD OPTIONS

-exportselection -headinglines -headings -hscroll -leftinfo -lefttitle -rightinfo -righttitle -selectmode -vscroll

WIDGET OPTIONS

-command
: Command to evaluate when the list contents change.

-updown
: Flag: if true, also display up and down buttons for moving table items vertically.

PUBLIC METHODS

clear {}
: Make the TableLists empty.

move_down {}
: Move the selected elements from the right to the left TableList.

move_left {}
: Move the selected elements from the right to the left TableList.

move_right {}
: Move the selected elements from the left to the right TableList.

move_up {}
: Move the selected elements from the right to the left TableList.

SEE ALSO

FrameWidget(n)

- - - - - -

Last change: 07 May 99

4.2.21 EntryForm(n)

NAME

EntryForm - Form dialog for entering data at given labels

NAMESPACE

util

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

EntryForm <path> ?options?

DESCRIPTION

An EntryForm widget is a dialog for entering data at given labels.

ITK COMPONENTS

buttons
: Frame containing dialog buttons.

canvas
: Canvas used to add a scrollbar.

entries
: Frame containing entries.

title
: Title for window.

WIDGET OPTIONS

-buttons
: List {{label cmd} {label cmd}} of additional buttons to display.

-command
: Called with list of values when Enter button is pushed.

-labels
: List of labels, one for each entry to be displayed.

-title
: Title for dialog.

-values
: Optional list of values, one for each entry to be displayed.

PUBLIC METHODS

cancel {}
: Called for the Cancel button.

reset {}
: Reset to the original values.

resize {frame canvas cw ch}

```
        Called when the window is resized. the arguments are the entries
        frame (in the canvas), the canvas and the canvas width and hight.

    set_entry {label value}
        Set (reset) the value for the given entry.
```

PROTECTED METHODS

```
    enter_data {}
        This method is called to enter the data when the Enter button is
        pressed. Call the command specified by the -command option with a
        list of values, one for each label.

    eval_cmd {cmd}
        Add the contents of this window as a list argument to the given
        command and then evaluate it.

    init {}
        Called after options have been evaluated.
```

PROTECTED VARIABLES

```
    entries_
        Array(label) of entry widget for given label.

    initialized_
        Flag: set to 1 after init.
```

SEE ALSO

```
    TopLevelWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.22 FileSelect(n)

NAME

FileSelect - OSF/Motif standard file selection dialog

NAMESPACE

util

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

FileSelect <path> ?options?

DESCRIPTION

This class implements a version of the OSF/Motif standard file selection dialog box using primitive widgets as the building blocks. This is a modified version of an old widget written by Mark Ulferts that is still used because of poor performance in the iwidgets2.x version. This should probably be replaced when we move to tcl8.x.

WIDGET OPTIONS

-button_1
The button label for button 1.

-button_2
The button label for button 2.

-button_3
The button label for button 3.

-dir
Initial directory, default to [pwd].

-dirlabel
Label for directory list, default "Directories".

-dispdir
Display directory list, yes or no, default yes.

-dispfile
Display file list, yes or no, default yes.

-dispfilter
Display filter, yes or no, default yes.

-dispselect
Display selection, yes or no, default yes.

-filelabel
Label for file list, default "Files".

-filter
File list filter, defaults to "*".

-filter_types

```
               Add an optional menu of file suffixes (preset file filters).

       -filterlabel
              The label string above the filter widget.

       -full
              Display full file names, yes/no, default no.

       -height
               Set the height of the directory and file lists.

       -modal
               flag: if true, grab the screen.

       -selectlabel
              Label for selection entry, default "Selection".

       -title
              Set the window title.

       -width
              Set the width of the selection and filter entry widgets.
```

PUBLIC METHODS

```
       activate {}
              Perform a grab operation, install the button callbacks, and wait
              for the result.  Make sure to reset the working directory back to
              the original before returning the result.

       get {}
               Return the selection.

       set_filter_type {type}

              Method: Set the filter type to a known value.
              .
```

PROTECTED METHODS

```
       _cancelcmd {}



- - - - - -
Last change: 07 May 99
```

4.2.23 FrameWidget(n)

NAME

FrameWidget - Itk base class for widgets with their own frame

NAMESPACE

util

PARENT CLASS

itk::Widget

SYNOPSIS

FrameWidget <path> ?options?

DESCRIPTION

The FrameWidget itcl class is a subclass of itk::Widget and thus inherits all of the features described in Widget(n). In addition, a number of useful methods are defined, for use by the derived classes.

PUBLIC METHODS

add_short_help {w text}
: Set the text of short help message to be displayed whenever the mouse enters the widget w (assumes you are using class ToplevelWidget).

busy {cmd}
: Run the given tcl command while displaying the busy cursor in the frame's parent top level window.

short_help {text}
: Set the text of the short help message (display now, assumes you are using class ToplevelWidget).

PROTECTED METHODS

init {}
: Derived classes can re-define the "init" method to run code after all options have been evaluated.

PROTECTED VARIABLES

class_
: Name of this (derived) class.

w_
: Shorter name for $itk_component(hull).

SEE ALSO

Widget(n)

- - - - - -

Last change: 07 May 99

4.2.24 GraphPrint(n)

NAME

GraphPrint - Print dialog box for printing the contents of a graph

NAMESPACE

util

PARENT CLASS

util::CanvasPrint

SYNOPSIS

GraphPrint <path> ?options?

DESCRIPTION

This widget defines a print dialog box for printing the contents of a graph.

WIDGET OPTIONS

-graph

Graph to print.

PUBLIC METHODS

ok {args}

Use PrintDialog::ok to get the fd (CanvasPrint::ok returns filename).

print {fd}

Print the contents of the graph as postscript to the given file descriptor.

SEE ALSO

CanvasPrint(n)

- - - - - -

Last change: 07 May 99

4.2.25 HelpWin(n)

NAME

HelpWin - Itcl class for displaying a text window with a help text.

NAMESPACE

util

SYNOPSIS

HelpWin <path> ?options?

DESCRIPTION

This is a simple class that displays the contents of a named file in a text widget. The file should be set as a configuration option when an instance is created. This class shares a text window amongst all its instances, this means that it overwrites any previous contents.

WIDGET OPTIONS

-file

The name of the file whose contents are to be displayed in the help text widget.

PUBLIC METHODS

display {}

This method causes the object to take control of the text widget and display the help text.

remove_help {}

Remove the help window.

PROTECTED VARIABLES

fonts_

Names of available fonts.

- - - - - -

Last change: 07 May 99

4.2.26 InputDialog(n)

NAME

InputDialog - Dialog to display a message and get input from the user

NAMESPACE

util

PARENT CLASS

util::DialogWidget

SYNOPSIS

InputDialog <path> ?options?

DESCRIPTION

An InputDialog is a dialog to display a message and get input from the user.

ITK COMPONENTS

entry

Tk entry widget for input.

PROTECTED METHODS

init {}

Called after options have been evaluated.

set_result {}

This method is redefined here to change the value that is returned from activate to be the contents of the entry widget.

SEE ALSO

DialogWidget(n)

- - - - - -

Last change: 07 May 99

4.2.27 LabelCheck(n)

NAME

LabelCheck - Itcl megawidget for choosing options.

NAMESPACE

util

PARENT CLASS

util::LabelWidget

SYNOPSIS

LabelCheck <path> ?options?

DESCRIPTION

LabelCheck is an itcl megawidget for choosing options using a label and radiobuttons (uses blt_table to arrange the buttons in rows and columns).

ITK COMPONENTS

table
: Tk frame containing checkbuttons.

$name
: One Tk component is created for each item in the list given by the -choice argument.

STANDARD OPTIONS

-anchor -background -borderwidth -foreground -relief -state -valuefont -valuewidth

WIDGET OPTIONS

-choice
: List of choices.

-cols
: Max number of columns to display (0 for unlimited) note: specify either -rows OR -cols, but not both.

-command
: Command to run when value is changed (new value is appended).

-orient
: Widget orientation: horizontal or vertical.

-rows
: Max number of rows to display (0 for unlimited) note: specify either -rows OR -cols, but not both.

-value
: List of choices to display as ON, all others are then OFF.

-variable
: Name of global variable to set (defaults to $w_.choice).

PUBLIC METHODS

clear {}
Clear the choice display.

get {}
Return the current value of this option.

itemconfig {name args}
Configure the individual buttons by name.

PROTECTED METHODS

do_layout {}
Layout the choices according to the options.

PROTECTED VARIABLES

but_
Array(name) of button widgets.

variable_
Name of text variable array for tracing.

SEE ALSO

LabelWidget(n)

- - - - - -
Last change: 07 May 99

4.2.28 LabelChoice(n)

NAME

LabelChoice - Widget for choosing options using a label and radiobuttons

NAMESPACE

util

PARENT CLASS

util::LabelWidget

SYNOPSIS

LabelChoice <path> ?options?

DESCRIPTION

LabelChoice is an itcl megawidget for choosing options using a label and radiobuttons (uses blt_table to arrange the buttons in rows and columns).

ITK COMPONENTS

table
: Tk frame containing radiobuttons.

choice$i
: One Tk component is created for each item in the list given by the -choice argument. The component names are choice0, choice1, ...

STANDARD OPTIONS

-anchor -background -borderwidth -foreground -relief -state -valuefont -valuewidth

WIDGET OPTIONS

-choice
: List of choices.

-cols
: Max number of columns to display (0 for unlimited) note: specify either -rows OR -cols, but not both.

-command
: Command to run when value is changed (new value is appended).

-orient
: Widget orientation: horizontal or vertical.

-rows
: Max number of rows to display (0 for unlimited) note: specify either -rows OR -cols, but not both.

-value
: Value to display.

-variable
: Name of global variable to set (defaults to $w_.choice).

PUBLIC METHODS

clear {}
Clear the choice display.

get {}
Return the current value of this option.

itemconfig {name args}
Configure the individual buttons by name.

PROTECTED METHODS

do_layout {}
Layout the choices according to the options.

PROTECTED VARIABLES

but_
Array(name) of button widgets.

variable_
Name of text variable for tracing.

SEE ALSO

LabelWidget(n)

- - - - - -
Last change: 07 May 99

4.2.29 LabelEntry(n)

NAME

LabelEntry - Itk widget for displaying a labeled entry

NAMESPACE

util

PARENT CLASS

util::LabelWidget

SYNOPSIS

LabelEntry <path> ?options?

DESCRIPTION

This widget displays a label and an entry and implements convenient methods for accessing and modifying the label and the value.

ITK COMPONENTS

entry

Tk entry widget.

STANDARD OPTIONS

-borderwidth -relief -show -textvariable -valuefont

WIDGET OPTIONS

-autoselect

Select the contents of the entry whenever it gets the focus.

-changecmd

Commands to evaluate whenever the entry value changes.

-command

The command for <Return> in the entry, called with the new value.

-entrycommand

Alternative name for use when -command is already used by a derived class.

-invalid

For compat with iwidgets entryfield: action for invalid char with -validate .

-justify

Set to "right" to make sure the end of the entry is visible.

-orient

Widget orientation: horizontal or vertical.

-state

Set the state to normal or disabled (greyed out).

-validate

Valiadation of entry fields (based on iwidgets entryfield class)

```
            numeric, alphabetic, integer, hexidecimal, real, and
            alphanumeric.

        -value
               Set the value displayed in the entry.

        -valuewidth
               Set the width of the value displayed.
```

PUBLIC METHODS

```
        get {}
                Get the value in the entry.

        select {}
               Select the contents of the entry.
```

PROTECTED VARIABLES

```
        notrace_
               This flag is set to 1 to avoid tracing when changing the entry's
               value.

        validate_cmd_
               Command to validate entry contents.
```

SEE ALSO

```
        LabelWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.30 LabelEntryScale(n)

NAME

LabelEntryScale - Widget combining a labeled entry with a scale widget

NAMESPACE

util

PARENT CLASS

util::LabelEntry

SYNOPSIS

LabelEntryScale <path> ?options?

DESCRIPTION

LabelEntryScale is an Itcl widget combining a labeled entry with a scale widget to make a scale with an editable entry field.

ITK COMPONENTS

left
: Left arrow button.

right
: Right arrow button.

scale
: Tk scale widget.

scaleframe
: Tk frame containing scale widget.

STANDARD OPTIONS

-background -digits -foreground -length -resolution -scaleWidth

WIDGET OPTIONS

-command
: Command to execute when the value changes.

-from
: Scale range -from.

-increment
: Amount to add or subtract for each button push.

-orient
: Widget orientation: horizontal or vertical.

-show_arrows
: Flag: if true, display left and right arrows for incrementing the value.

-state
: Set the state to normal or disabled (greyed out).

```
-to
       Scale range -to.

-value
       Set the value in the entry.
```

PROTECTED METHODS

```
increment {sign}
       Increment (1) or decrement (-1) the value by the current
       increment.

scaleCmd {newValue}
       This method is called for changes in the scale widget.  It does
       nothing unless the value has changed, since the Scale has a motion
       event that is constantly being invoked.

start_increment {sign}
        Start incrementing if not already doing so.

stop_increment {}
        Stop incrementing.
```

PROTECTED VARIABLES

```
afterId_
        State of increment command.

from_
       Original -from and -to values.
```

SEE ALSO

```
LabelEntry(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.31 LabelMenu(n)

NAME

LabelMenu - Itcl widget for displaying a label and a menubutton

NAMESPACE

util

PARENT CLASS

util::LabelWidget

SYNOPSIS

LabelMenu <path> ?options?

DESCRIPTION

This widget displays a label and a menubutton with a selector and a menu of radiobuttons. This can be used for choosing items from a list and displaying the current choice. The widget supports adding of items to the radiobutton menu and keeps track of which items are selected.

ITK COMPONENTS

mb

Menubutton containing the menu.

menu

Component for the menu.

STANDARD OPTIONS

-borderwidth -indicatoron -relief -state -valueanchor -valuefont -valuewidth

WIDGET OPTIONS

-orient

Widget orientation: horizontal or vertical.

-value

Set the selected value (referenced by label or bitmap) note: this.var is the trace variable and values_ holds the valid values.

-variable

Global variable linked to menu.

PUBLIC METHODS

add {args}

Add an item to the menu. The args may be the options: -label <label for menuitem and menubutton when chosen> -bitmap <bitmap for menuitem and menubutton when chosen> -command <cmd to execute when item is selected> -background <color of menu item and button when chosen> -font <font of menu item and button when chosen>.

add_separator {}

```
            Add a separator item to the menu.

        clear {}
            Remove all of the items in the menu.

        contains {name}
            Return true if the LabelMenu contains the given item (specified by
            the label or bitmap name).

        get {}
            Return the current value.

        update_menubutton {args}
            Update the label on the menubutton.
```

PROTECTED METHODS

```
        variable_changed {args}
            Called when "$variable_" has changed, via trace, to update menu.
```

PROTECTED VARIABLES

```
        default_bg_
            Default background color.

        values_
            Array(label or bitmap name) of values for selecting a
            radiobutton.

        variable_
            Trace variable name.
```

SEE ALSO

```
        LabelWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.32 LabelMessage(n)

NAME

LabelMessage - Itcl widget for displaying a label and a message

NAMESPACE

util

PARENT CLASS

util::LabelWidget

SYNOPSIS

LabelMessage <path> ?options?

DESCRIPTION

LabelMessage is an Itcl widget for displaying a label and a message.

ITK COMPONENTS

message

Tk message component.

STANDARD OPTIONS

-anchor -aspect -background -borderwidth -foreground -justify
-relief -textvariable -value -valuefont -valuewidth

WIDGET OPTIONS

-orient

Widget orientation: horizontal or vertical.

PUBLIC METHODS

get {}

Get the value in the message.

SEE ALSO

LabelWidget(n)

- - - - - -
Last change: 07 May 99

4.2.33 LabelNumber(n)

NAME

LabelNumber - Widget displaying a label, a number, and arrow buttons

NAMESPACE

util

PARENT CLASS

util::LabelValue

SYNOPSIS

LabelNumber <path> ?options?

DESCRIPTION

LabelNumber is an Itcl widget for displaying a label, a number and buttons to increment and decrement the number.

ITK COMPONENTS

bframe
: Button frame.

decr
: Button to decrement number.

incr
: Button to increment number.

STANDARD OPTIONS

-background -foreground -state

WIDGET OPTIONS

-command
: Commands to evaluate whenever the entry value changes.

-increment
: Amount to add or subtract for each button push.

-max
: Maximum value.

-min
: Minimum value.

-orient
: Widget orientation: horizontal or vertical.

PROTECTED METHODS

increment {sign}
: Increment (1) or decrement (-1) the value by the current increment.

SEE ALSO

LabelValue(n)

- - - - - -

Last change: 07 May 99

4.2.34 LabelValue(n)

NAME

LabelValue - Widget displaying a label and a selectable value

NAMESPACE

util

PARENT CLASS

util::LabelEntry

SYNOPSIS

LabelValue <path> ?options?

DESCRIPTION

This widget displays a label and a value (also a label(n)) and implements convenient methods for accessing and modifying the label and the value.

SEE ALSO

LabelEntry(n)

- - - - - -
Last change: 07 May 99

4.2.35 LabelWidget(n)

NAME

LabelWidget - Base class of labeled widgets

NAMESPACE

util

PARENT CLASS

util::FrameWidget

SYNOPSIS

LabelWidget <path> ?options?

DESCRIPTION

LabelWidget is an itcl widget for displaying a label. Another widget may be added in a derived class...

ITK COMPONENTS

label

Label for the widget.

STANDARD OPTIONS

-anchor -background -foreground -labelfont -labelwidth -text

WIDGET OPTIONS

-disabledforeground

Disabled foreground color.

-orient

Widget orientation: horizontal, vertical.

-state

Set the state to normal or disabled (greyed out).

PROTECTED VARIABLES

side_

Pack option.

SEE ALSO

FrameWidget(n)

- - - - - -

Last change: 07 May 99

4.2.36 ListboxWidget(n)

NAME

ListboxWidget - Widget for scrolled lists, based on the Tk listbox

NAMESPACE

util

PARENT CLASS

util::FrameWidget

SYNOPSIS

ListboxWidget <path> ?options?

DESCRIPTION

ListboxWidget is an itcl widget for scrolled lists, based on the Tk listbox.

ITK COMPONENTS

hscroll
: Optional horizontal scrollbar.

listbox
: Tk listbox.

title
: Optional title.

vscroll
: Optional vertical scrollbar.

STANDARD OPTIONS

-borderwidth -exportselection -font -height -relief -selectmode -titlefont -width

WIDGET OPTIONS

-editable
: Enable/disable editing of rows.

-hscroll
: Flag: if true, list will have a horizontal scrollbar.

-title
: Title string for listbox.

-vscroll
: Flag: if true, list will have a vertical scrollbar.

PUBLIC METHODS

append {line}
: Append a line to the list.

append_list {list}

```
       Append a list of items to the list.

clear {}
       Make the list empty.

clear_selection {}
       Clear the selection.

deselect_row {n}
       Deselect the row with the given index.

edit_row {}
       Pop up a dialog window to edit the value in the selected row.

get_contents {}
       Return the contents of the listbox as a tcl list.

get_selected {}
       Return a list of the selected items.

move_down {}
       Move the selected row down 1 row.

move_up {}
       Move the selected row up 1 row.

num_selected {}
       Return the number of rows currently selected in the table.

remove_selected {}
       Remove the selected items from the listbox and return them as a
       list.

select_row {n {clear 1}}
       Select the row with the given index. If clear is 1, clear the
       selection first.

select_rows {from to}
       Select a range of rows.

set_contents {list}
       Set the contents of the listbox from the list.

set_row {oldrow newrow}
       Replace the contents of the given row with the new row Note: this
       assumes that no 2 rows are exactly alike.

show_last_row {}
       Scroll to the end of the table and display the last set of rows.
```

PROTECTED VARIABLES

```
listbox_
       Listbox widget.
```

SEE ALSO

```
FrameWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.37 ListDialog(n)

NAME

ListDialog - Dialog to display a listbox window and some text

NAMESPACE

util

PARENT CLASS

util::DialogWidget

SYNOPSIS

ListDialog <path> ?options?

DESCRIPTION

A ListDialog is a dialog to display a listbox window and some text.

ITK COMPONENTS

list

Listbox widget below the message.

STANDARD OPTIONS

-listboxfont -listboxheight -listboxwidth

WIDGET OPTIONS

-choice

List of items to choose from.

PROTECTED METHODS

set_result {}

This method may be redefined in a derived class to change the value that is returned from activate (default is the number of the button selected).

SEE ALSO

DialogWidget(n)

- - - - - -

Last change: 07 May 99

4.2.38 PasswdDialog(n)

NAME

PasswdDialog - Dialog to display a message and get a username

NAMESPACE

util

PARENT CLASS

util::DialogWidget

SYNOPSIS

PasswdDialog <path> ?options?

DESCRIPTION

A PasswdDialog is a dialog to display a message and get a username and password from the user.

ITK COMPONENTS

passwd
: LabelEntry widget for the password.

username
: LabelEntry widget for the username.

PROTECTED METHODS

init {}
: Called after options have been evaluated.

set_result {}
: This method is redefined here to change the value that is returned from activate to be the contents of the entry widget.

SEE ALSO

DialogWidget(n)

- - - - - -
Last change: 07 May 99

4.2.39 PrintDialog(n)

NAME

PrintDialog - Base class of Popup dialogs for specifying printer options

NAMESPACE

util

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

PrintDialog <path> ?options?

DESCRIPTION

PrintDialog is the base class of Popup dialogs for specifying printer options.

ITK COMPONENTS

btns
: Row of buttons (ButtonFrame(n)).

config
: Frame for misc other items added by sublasses (child-site...).

print_to_file
: Print to file entry (CheckEntry(n)).

printcmd
: LabelEntry for print command.

ptf
: Frame for print to file entry.

top
: Frame for printer name entry.

WIDGET OPTIONS

-filename
: File name for print to file.

-print_to_file
: Bool: print to file ?.

-printcmd
: Default print command.

-suffix
: Default suffix for print to file.

-userfile
: User defaults file (without path, $HOME will be added).

PROTECTED METHODS

```
file_writable {file}
       This method checks if a file can be used for writing.  Returns the
       'globbed' filename if so otherwise an empty string.

get_pathname {file}
       Return 'globbed' pathname for a file. If the extension is not set
       in $file then $itk_option(-suffix) is used.

init {}
       This method is called after all options have been evaluated.

load {}
       Load the user config file (created by save) if one exists,
       otherwise set the default values. The file is searched for in
       $HOME and the current dir.

ok {args}
       This method is called when the OK button is pressed Open the file
       or pipe and pass the fd to the subclass print method.

print {fd}
       Should be redefined in subclass.

save {}
       This method is called when the save button is pressed to save the
       print configuration for this user in a file for later loading.
```

PROTECTED VARIABLES

```
magic_
       Header string for userfile for comparison.
```

SEE ALSO

```
TopLevelWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.40 ProgressBar(n)

NAME

ProgressBar - widget to display a message and a progress bar

NAMESPACE

util

PARENT CLASS

util::FrameWidget

SYNOPSIS

ProgressBar <path> ?options?

DESCRIPTION

This object displays a message and a scale indicating that work is in progress.

ITK COMPONENTS

entry
: Entry widget used to display the progress message.

scale
: Scale widget used to show progress.

STANDARD OPTIONS

-font -length -width

WIDGET OPTIONS

-busycolor
: Color of bar when busy.

-from
: Set the starting point for the scale.

-idlecolor
: Color of bar when busy.

-sliderlength
: Length of slider (displayed when busy).

-speed
: Speed in ms/tick when looking busy.

-text
: Set the text of the label.

-to
: Set the end point for the scale.

-value
: Set the value of the bar between from and to.

PUBLIC METHODS

look_busy {{bool 1}}
Start (or stop) some animation to indicate that work is in progress. This will continue until the next reset or until -to is set with config.

reset {{text ""}}
Reset the bar to the beginning.

PROTECTED METHODS

do_something {}
Do something to look busy.

PROTECTED VARIABLES

color_
Current bar color.

inc_
Controls direction for busy animation.

looking_busy_
Flag: true if currently looking busy.

SEE ALSO

FrameWidget(n)

- - - - - -
Last change: 07 May 99

4.2.41 ScrollText(n)

NAME

ScrollText - Itcl class for displaying a text window with scrollbars.

NAMESPACE

util

PARENT CLASS

FrameWidget

SYNOPSIS

ScrollText <path> ?options?

DESCRIPTION

This class defines methods and configuration options for creating a text widget with scrollbars. The scrolltext widget has scrollbars along the right and bottom.

ITK COMPONENTS

Corner
: Corner frame.

Frame
: Outer frame.

Label
: Tk label for title.

Scrollbottom
: Horizontal scrollbar.

Scrollright
: vertical scrollbar.

Text
: Tk text widget.

STANDARD OPTIONS

-exportselection -font -height -width

WIDGET OPTIONS

-label
: Adds a label over at top of the text widget.

PUBLIC METHODS

clear {args}
: Clears a range of items from the text widget. If first is "all" then all lines are deleted. If only first is given then this clears a single line. "last" may be set as end.

get {index}
: Gets the item with the given indices from the text widget.

```
       insert {index args}
              Inserts a line of text with the given index. "index" can be 0 or
              end which inserts at the beginning and at the end.
```

SEE ALSO

```
       FrameWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.42 TableList(n)

NAME

TableList - A listbox based table widget

NAMESPACE

util

PARENT CLASS

util::ListboxWidget

SYNOPSIS

TableList <path> ?options?

DESCRIPTION

TableList is an itcl widget for displaying tabular information and headings in a Tk listbox. It lines up columns of data (specified in tcl list format) with column headings, optionally sorts the data by given columns, in a given order, can "hide" columns, rearrange columns, or display columns matching certain expressions. You can even print the contents of the table. This widget has been in use for a number of years and has grown to include quite a few features. It is well suited to displaying the results of database or catalog queries.

ITK COMPONENTS

config_file
: This class creates an object for reading and writing an optional config file and managing table options.

headbox
: Listbox used to display table headings.

menubutton
: Optional menu with table operations.

STANDARD OPTIONS

-borderwidth -exportselection -headingfont -headinglines -relief -selectmode -width

WIDGET OPTIONS

-filtercmd
: Command to call to filter each row, hook to modify the row before it is displayed arg is is the name of the list holding the row (call by reference).

-formats
: List of printf formats for columns (if not specified, will be calculated, see also -sizes).

-headings
: Field names for heading - Note: specify before "-info".

-hformats
: Print format string for the headings (set after -formats, defaults

```
        to same as $formats_) This might be different for headings if a
        column uses %f formats...

    -ignore_case
        Flag: if true, ignore case in matching (only works when
        -use_regexp 1 was specified).

    -info
        List of lists, one per line to display in table/list.

    -layoutcommand
        Command to call when layout options have been selected.

    -menubar
        Name of menubar frame in which to place table menubutton
        (optional).

    -order
        Set the order of the columns.

    -printcmd
        Default print command.

    -sizes
        List of column sizes (if not specified, will be calculated).

    -sort_by
        List of col index: sort table based on given columns (empty means
        don't sort).

    -sort_cols
        List of col names to sort by (empty means don't sort).

    -sort_order
        Set direction of sort: may be one of (increasing, decreasing) .

    -sortcommand
        Command to call when sort options have been selected.

    -static_col_sizes
        If true, reuse the calculated column sizes rather than recalculate
        for new info.

    -use_regexp
        Flag: if true, use regular exprs for matching, otherwise use
        wildcards.
```

PUBLIC METHODS

```
    add_row {newrow}
        Add a new row to the list and update the display.

    append_row {row}
        Append a row to the table. (call new_info when done).

    append_rows {rows}
        Append a list of rows to the table.

    calculate_format {}
        Calculate the print formats for table rows from the max column
        widths and options. The format string takes care of column widths,
        show/hide column and column separators.
```

```
       If the format was set explicitly (formats_flag_ = 1), use it
       otherwise, if the column widths are known (-sizes was set) use
       them, otherwise calculate the max column widths from the info
       list.

       The formats list uses %n$-s type format strings to set left/right
       alignment, width and order all at once, for example:  "%-10s" for
       left justify, "%.0s" effectively hides the item...

       If the Precision option is set for a column, assume it is a
       floating point value to be formatted like: %6.2f for example
       (precision = 2).

clear {}
       Make the table empty.

edit_row {}
       Pop up a dialog window to edit the values in the selected row.

get_contents {}
       Return the contents of the table as a list of rows.

get_headings {}
       Return the table headings.

get_option {name option}
       Return the option value for the given heading name.  See above for
       list of Options...

get_selected {}
       Return a list of the selected rows note: use disp_info_, since
       listbox contains formated lines.

get_selected_with_rownum {}
       Return a list of {{rownum row} {rownum row} ...} for the selected
       rows.

info_rows {}
       Return the number of rows being displayed in the table (after
       matching).

layout_dialog {}
       Pop up a window to change the layout of the table and call the
       optional command when done.

make_table_menu {}
       Add the table config menu items to the given menu.

move_down {}
       Move the selected row down 1 row and make the changes in the info
       list.

move_up {}
       Move the selected row up 1 row and make the changes in the info
       list.

new_headings {}
       This method is called whenever the headings list changes.

new_info {}
       This method is called whenever the info list changes.

print {fd}
```

```
       Print the contents of the table to the open file descriptor.

print_dialog {}
       Pop up a dialog to print the contents of the table to a printer or
       file.

remove_row {row}
       Remove the given row (given by its value).

remove_selected {}
       Remove the selected rows from the table and return them as a list
       of lists.

restore_selection {}
       Restore the previously saved row selection.

restore_yview {}
       Restore the previously saved scroll position .

save_dialog {}
       Get a name from the user and use it to save the current
       configuration to a file under the user's home directory.

save_selection {}
       Save a list of the currently selected rows so they can be restored
       later. .

save_yview {}
       Save the current scroll position so it can be restored later. .

search {name value}
       Search for and highlight the first row containing the given value
       in the named column.

set_option {name option value}
       Set the option value for the given heading name

       Options:
        Show (bool)         - display or don't display the column Align
        (Left,Right)  - for left or right justify Separator           -
        set the separator string (goes after col) Wildcard            -
        only show rows where wildcard matches Precision           -
        number of places after the decimal for floating point values.

set_options {headings option value}
       Same as set_option, but works on a list of column heading names.

set_row {oldrow newrow}
       Replace the contents of the given row with the new info Note: this
       assumes that no 2 rows are exactly alike.  We can't use the row
       index here, since sorting and matching may mix things up too
       much.

sort_dialog {}
       Pop up a dialog to sort the contents of the table.

total_rows {}
       Return the total number of rows (before matching).

update_sort_info {}
       Update the indexes ($sort_by) for the sort columns.

xview {args}
```

```
        Scroll both the heading box and the main listbox syncronously
        (called for horizontal scrolling in listbox).

    yview {args}
        Scroll the listbox vertically.
```

PROTECTED METHODS

```
    delete_config {file}
        Get a name from the user and use it to save the current
        configuration to a file under the user's home directory.

    load_config {file}
        Load the named config file and update the display based on the new
        settings.
```

PROTECTED VARIABLES

```
    disp_info_
        List of info to display (after filtering and sorting).

    formats_
        Printf format string for table rows.

    formats_flag_
        Flag: true if the -formats option was specified so that we don't
        have to calculate the format string for a row.

    headbox_
        Box for headings.

    headings_
        Table column headings.

    hformats_
        Printf format string for table headings.

    hsize_
        Array(col) of heading width.

    info_
        Table contents as list of rows/cols.

    info_cols_
        Number of columns in table .

    info_rows_
        Number of rows in the info list (not including hidden rows).

    line_length_
        Length of a line in the table.

    match_all_
        Flag: true if match_list_ should match all rows.

    match_any_
        String used to match any string.

    match_list_
        List of glob expressions for matching rows to wildcards.

    match_proc_
        Method to use for matching rows (match_glob_ or match_regexp_).
```

```
menubutton_
       Menubutton widget.

num_cols_
       Number of columns.

ord_
       Array(heading) of column order.

order_
       List of indexes in row for headings (indep. of viewing order).

saved_selection_
       Used for save_/restore_selection methods.

saved_yview_
       Used for save_/restore_yview methods.

size_
       Array(col) of col width.

total_rows_
       Total number of rows, including hidden rows.
```

SEE ALSO

ListboxWidget(n)

- - - - - -
Last change: 07 May 99

4.2.43 TableListConfig(n)

NAME

TableListConfig - Popup widget for editing table layout for TableList

NAMESPACE

util

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

TableListConfig <path> ?options?

DESCRIPTION

TableListConfig is an itcl popup widget for editing table layout for the TableList class.

ITK COMPONENTS

btns
: Row of buttons (ButtonFrame(n)).

dlist
: DoubleList widget for choosing columns to show or hide.

WIDGET OPTIONS

-command
: Command to eval when done.

-table
: TableList whose layout is being edited.

PUBLIC METHODS

cancel {}
: Called for the cancel button - reset to original values and quit.

deselect_all {}
: Deselect all items in the lists and disable the entries and options.

load {}
: The information was read by $config_file_ already, so we just display it here.

ok {args}
: Accept the current information.

reset {}
: Reset to default configuration.

PROTECTED METHODS

init {}
: Initialize layout .

```
left_selected {}
       Called when an item in the left listbox is selected.

make_button_row {}
       Make the row of buttons at bottom.

make_list_window {}
       Create a window with 2 scrolling lists and arrows between then to
       move items back and forth.

right_selected {}
       Called when an item in the right listbox is selected.
```

PROTECTED VARIABLES

```
config_file_
       Table config class.

leftbox_
       Left and right listboxes.

options_
       Array(name,option) of option values, loaded from config_file.

table_
       TableList whose layout is being edited.
```

SEE ALSO

```
TopLevelWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.44 TableListConfigFile(n)

NAME

TableListConfigFile - Class for managing config info for TableList widget

NAMESPACE

util

PARENT CLASS

util::FrameWidget

SYNOPSIS

TableListConfigFile <path> ?options?

DESCRIPTION

TableListConfigFile is an itcl class (no windows) for managing the config file for the TableList widget. See also TableListConfig(n) for the user interface to this class.

WIDGET OPTIONS

-headings
Column headings from table.

-order
List of headings, in the order they (and the columns) should be displayed.

-sort_by
List of col index: sort table based on given columns (empty means don't sort).

-sort_order
Set direction of sort: may be one of (increasing, decreasing) .

-sort_type
Set type of sort key: may be one of (ascii, integer, real) .

-use_regexp
Flag: if true, use regular exprs for matching, otherwise use wildcards.

PUBLIC METHODS

get_option {name option}
Return the value of the given option for the given heading name (see comments above for set_option...).

get_options {ar}
Write the current option values to the named array.

get_order {}
Return a list of column headings in the order they are displayed.

load {file}
Load the user config file (created by save) if one exists, otherwise set the default values. .

```
save {file}
       Save the current information to the given file for later loading
       (with "load" below).

set_defaults {{force 0}}
       Set the default values in case there is no config file or it was
       out of date

       If force is 1, set even values that are already set.

set_option {name option value}
       Set the option value for the given heading name

       Options:
        Show (bool)         - display or don't display the column Align
        (Left,Right)  - for left or right justify Separator           -
        set the separator string (goes after col) Wildcard            -
        only show rows where wildcard matches Precision           -
        number of places after the decimal for floating point columns.

set_options {ar}
       Read option values from the named array.
```

PROTECTED VARIABLES

```
headings_
       Table column headings.

initialized_
       Set to 1 after initialization.

magic_
       Magic string for config files to keep track of versions.

match_any_
       String used to match any string.

options_
       Array(heading,opt) of option values.

order_
       Sort order.

saved_headings_
       Previous column headings - read from config_file.

sort_by_
       Sort keys.

sort_order_
       Direction of sort: may be one of (increasing, decreasing) .

sort_type_
       Set type of sort key: may be one of (ascii, integer, real) .
```

SEE ALSO

```
FrameWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.45 TableListPrint(n)

NAME

TableListPrint - Print dialog box for printing the contents of a TableList

NAMESPACE

util

PARENT CLASS

util::PrintDialog

SYNOPSIS

TableListPrint <path> ?options?

DESCRIPTION

TableListPrint is a print dialog box for printing the contents of a TableList.

ITK COMPONENTS

cols
: LabelCheck(n) widget displaying table columns to print.

options
: Print options frame.

WIDGET OPTIONS

-choose
: Flag: if true, add a widget for choosing the columns to print.

-table
: Handle of the TableList class that created this dialog (used to access the current values there...).

PUBLIC METHODS

print {fd}
: This method is redefined here from the parent class to print the contents of the TableList with specified columns.

PROTECTED METHODS

init {}
: This method is called after all options have been evaluated.

make_choice_frame {}
: Add a frame to let the user choose which columns to print.

PROTECTED VARIABLES

headings_
: Table column headings.

SEE ALSO

PrintDialog(n)

```
- - - - - -
Last change: 07 May 99
```

4.2.46 TableListSort(n)

NAME

TableListSort - itcl popup widget for sorting contents of a TableList

NAMESPACE

util

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

TableListSort <path> ?options?

DESCRIPTION

TableListSort is an itcl popup widget for sorting contents of a TableList.

ITK COMPONENTS

dlist

DoubleList(n) widget to display sort keys to use or not use.

WIDGET OPTIONS

-command

Command to eval when sort options have been selected. called with 2 args: 1: list of sort cols, 2: sort order (increasing or decreasing).

-table

TableList whose layout is being edited.

PUBLIC METHODS

apply {}

Apply the changes (called for OK button).

cancel {}

Called for the cancel button.

load {}

Load the sort settings from the config file object.

reset {}

Reset the sorting parameters to the default values.

table_sort {sort_cols sort_order}

Default method called when sort options have been selected. May be overridden with the -command option. sort_cols is a list of column names to sort by sort_order is one of "increasing", "decreasing".

PROTECTED METHODS

create_main_window {}

Do the main window layout.

```
        init {}
               Initialize components.

        make_button_row {}
               Make the row of buttons at bottom.

        make_dlist_window {}
               Create a window with 2 lists - one for the headings and one for
               the sort keys.

        make_options_window {}
               Create a window for setting column options.
```

PROTECTED VARIABLES

```
        config_file_
               Object managing config info.

        table_
               TableList widget.
```

SEE ALSO

```
        TopLevelWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.47 TextDialog(n)

NAME

TextDialog - Dialog widget displaying a text window

NAMESPACE

util

PARENT CLASS

util::DialogWidget

SYNOPSIS

TextDialog <path> ?options?

DESCRIPTION

TextDialog is a dialog widget for displaying a text window and a specified text.

ITK COMPONENTS

textf
: Frame for text widget.

textwin
: Tk text widget.

vscroll
: Vertical scrollbar.

STANDARD OPTIONS

-textfont -textheight -textstate -textwidth

WIDGET OPTIONS

-contents
: Contents of the text window.

SEE ALSO

DialogWidget(n)

- - - - - -

Last change: 07 May 99

4.2.48 TopLevelWidget(n)

NAME

TopLevelWidget - Itk base class for popup windows

NAMESPACE

util

PARENT CLASS

itk::Toplevel

SYNOPSIS

TopLevelWidget <path> ?options?

DESCRIPTION

The TopLevelWidget itcl class is a subclass of itk::Toplevel and thus inherits all of the features described in Toplevel(n). In addition, a number of useful methods are defined, for use by the derived classes.

ITK COMPONENTS

menubar
: Optional menubar added to top of window.

short_help
: Optional short help window at bottom.

$name
: Menubutton component. $name is the same as the menubutton label, but in lower case.

STANDARD OPTIONS

-helpfont

WIDGET OPTIONS

-center
: If true, center the application window on startup.

-number
: Set an optional unique instance or clone number for this window. If this is not set, it is taken from the main window name, which should be ".<appname><number>".

-shorthelpwin
: Optionally specify a different TopLevelWidget to display short help messages.

-standalone
: True if running as separate process.

-transient
: If true and this is a child of another TopLevel widget, make the toplevel window transient - so that it will open and close with its parent.

```
-withdraw
       If true, withdraw the application window on startup.
```

PUBLIC METHODS

```
add_help_button {file label {msg ""}}
       If any help has been defined then add a button to show it.

add_menu_short_help {menu label msg}
       Set the text of the short help message to be displayed whenever
       the mouse enters the menu item with the given label (see
       short_help below).

add_menubar {}
       Add a menubar to the main frame.

add_menubutton {label {helptext ""} {side left}}
       Add a menu button to the menubar, or reset it's menu to empty if
       the menubutton already exists.  If helptext is specified, it is
       used as the short help text.  If side is specified, the menu
       button is placed on the given side.  The return value if the name
       of the menubutton's menu.

add_menuitem {menu type label msg args}
       Add a menu item of the given type to the given menu and arrange to
       have the given short help message displayed when the mouse is over
       the item.  The extra args are passed to the "$menu add" command.

add_short_help {w msg}
       Set the text of the short help message to be displayed whenever
       the mouse enters the widget w (see short_help below) If "win" is
       specified, it is used to display the help text rather than this
       window (it should also be a TopLevelWidget).

busy {cmd}
       Run the given tcl command in the scope of this class while
       displaying the (blt) busy cursor in the toplevel window.

center_window {}
       Center this window on the screen.  this doesn't work with gridded
       geometry...

configure_menubutton {label args}
       Configure something of the named menubutton.

get_menu {label}
       Return the path name of the menu for the given menubutton label.

get_menubutton {label}
       Return the path name of the menubutton for the given menubutton
       label.

hide_windows {variable}
       Toggle the visibility of all popup windows (except this one).  The
       trace variable name is passed as an argument here. if 1, hide the
       windows, otherwise restore them again.

insert_menuitem {menu index type label msg args}
       Insert a menu item of the given type in the given menu before the
       given index and arrange to have the given short help message
       displayed when the mouse is over the item. The extra args are
       passed to the "$menu insert" command.
```

```
list_windows {{w .}}
       Return a list of top level windows that are children of $w.

make_short_help {{side bottom}}
       Add a subwindow at the bottom of the screen for short help
       messages.

quit {}
       Use this method to quit the application if you might want to reuse
       the window later.

show_help {file}
       Show the help file associated with this window.

test {cmd}
       Run the given tcl command and print out any errors.
```

PROTECTED METHODS

```
call_init {}
       Call the init method and then check if there is a plugin for this
       class.

call_plugin_procs {classname}
       Call each of the plugin procs for the given class, as found by the
       check_plugins proc above, passing the the name of this instance as
       an argument. $classname should be the name of this class.

init {}
       This method is called after all options have been evaluated and is
       meant to be redefined in a derived class.

menu_motion {m y}
       Called for motion events in menubar menus for short help use.

setup_menuitem {menu label msg args}
       Local method to set up the menu accelerator, help text, and
       command bindings.

short_help {msg {mf ""}}
       Set the text of the short help message (display now) Note:
       embedded bitmaps can be specified as follows:

       short_help "some text {bitmap mybitmap} other text ..."

       The optional mf arg is the "%m" value for the Enter/Leave event.
```

PROTECTED VARIABLES

```
bitmap_bg_
       Bitmap colors for short help area.

class_
       Name of this (derived) class.

help_button_menu_
       Name of help button.

help_window_
       Help window.

menu_item_help_
       Array(menu,menu-label) of help text for menu items.
```

```
popup_windows_
       Saved list of popup window names for "hide_windows" method.

short_help_win_
       Optional TopLevelWidget to display short help text (default:
       $this).

w_
       Shorter name for $itk_component(hull).
```

COMMON CLASS VARIABLES

```
checked_plugins_
       Array(classname) of bool: true if we checked for plugins already
       for the class.

clone_cnt_
       Clone number of this instance (for main windows).

command_
       Optional tcl command to eval for each TopLevelWidget created.

main_windows_
       Array(name) of main, application level top level windows.

plugin_procs_
       Array(classname) of list of plugin procs to call for each
       instance.

sourced_plugin_file_
       Array(filename) of bool: true if we sourced the plugin file.
```

SEE ALSO

```
Toplevel(n)
```

```
- - - - - -
Last change: 07 May 99
```

5 Installing the Tclutil Package

5.1 Before you build the Tclutil Package...

Make sure you have a proper Itcl-2.2 distribution (Tcl, Tk, BLT, TclX and ITCL extensions) with the necessary patches applied. Tclutil requires the following software to be already installed (not included):

- itcl-2.2 (includes tcl7.6, tk4.2)
- BLT-2.1
- tclX-7.6.0

These packages are available from the TCL archives.

See: http://www.tcltk.com/ for Itcl.

and http://www.NeoSoft.com/tcl/ for TclX and other contributed Tcl software

You can also get a copy of the whole Tcl/Tk source tree, with the patches already applied from ftp://ftp.archive.eso.org/pub/skycat/.

5.2 Build the Tclutil Package

To make the Tclutil package, configure and make as follows:

```
configure
make
make install
```

The default install dir is /usr/local. You can specify the -prefix argument to configure to change this:

```
configure -prefix $INSTALLDIR
```

If the environment variable TCLTK_ROOT is set, it is used as the default top level directory fo Tcl/Tk.

The default compilers used are g++ and gcc. If you wish to use another compiler, such as CC and cc, do this:

```
setenv CC cc
setenv CXX CC
configure -prefix $INSTALLDIR --with-cc
```

If you prefer using shared libraries and loadable Tcl modules (see 4) configure with:

```
configure --enable-shared
```

Note: If you are using g++, you must also have libg++ compiled as a shared library for this to work (libg++-2.7.2 also has the "--enable-shared" option) (This requires gcc-2.7.2.1 or newer on HP-UX).

5.3 VLT Make Procedure

As an alternative to running configure and make, you can also do this:

```
cd src
make
make install
```

The Makefile in the $TCLUTIL/src directory runs configure and then make as described above. You can also specify options to that Makefile, for example:

```
cd src
```

```
make PREFIX=$INSTALLDIR CONFIGURE_FLAGS=--with-gcc
```

The PREFIX variable defaults to /usr/local and is the prefix of the directory in which to install the software.

If the environment variable TCLTK_ROOT is set, it is used as the default top level directory fo Tcl/Tk.

5.4 Start the demo application

To run the demo, type:

```
INSTALLDIR/lib/tclutil/demos/tclutil
```

This package does not currently include any applications, so this just starts tclutil_wish with the correct environment. See the tclutil/test dir for short examples.

5.5 If you are using shared libraries

The Tclutil shared library libtclutil.sl (in HPs) or libtclutil.so (on Suns) is built with the same options used to build the Tcl shared library. The options are read from the file tclConfig.sh, which is searched for in the following places:

```
$prefix/lib/itcl          # $prefix is the value of the -prefix
$prefix/lib               # configure option, default: /usr/local
$TCLTK_ROOT/lib/itcl
$TCLTK_ROOT/lib
/vlt/tcltk/lib/itcl
/vlt/tcltk/lib
/usr/local/lib/itcl
/usr/local/lib
```

This assumes that you have built and installed Tcl with the same “--enable-shared” and compiler option used for Tclutil.

You may need to modify the SHLIB_PATH (HP) or LD_LIBRARY_PATH (Sun) environment variable so that the necessary shared libraries are found at run time. Both variables have the same format: a colon “:” separated list of directories to search for shared libraries.

From a tcl script you can load the TCLUTIL library dynamically with the command “load <path>/libtclutil.sl” or “load <path>/libtclutil.so” or it can be loaded automatically as a package. See the Tcl man pages for more information.

5.6 Tip for HP-UX users

If you are compiling under HP-UX, you can check with the program below if the shared library is loadable. If libtclutil.sl on HP-UX is not properly built you get the misleading error message “Not enough memory”.

```
#include <dl.h>
#include <errno.h>
#include <stdio.h>
main(int argc, char *argv[])
{
  shl_t handle;
  handle = shl_load(argv[1], BIND_IMMEDIATE | BIND_VERBOSE, 0L);
  if (handle == 0) {
    printf(“shl_load failed %s\n”, argv[1]);
    perror(“”);
  }
```

```
  else
    printf("shl_load ok\n");
}
```