NAME

container − Widget to contain a foreign window.

SYNOPSIS

container *pathName* ?*options*?

DESCRIPTION

The **container** widget lets you swallow another X11/Win32 toplevel or embed an X11 window from a foreign application into your Tk application. The foreign window is reparented inside of the widget. You can then place and arrange the container just as you would any Tk widget.

INTRODUCTION

Notebooks are a popular graphical paradigm. They allow you to organize many windows in a single widget. For example, you might have an application the displays several X-Y graphs at the same time. Typically, you can't pack the graphs into the same **frame** because they are too large. The other alternative is to pack the graphs into several **toplevel** widgets, allowing them to overlap on the screen. The problem is that all the different toplevel windows clutter the screen and are difficult to manage.

The **container** widget lets organize your application by displaying each graph as a page in a folder of a notebook. Only one page is visible at a time. When you click on a tab, the folder (graph) corresponding to the tab is displayed in the **container** widget. The container also lets you temporarily tear pages out of the notebook into a separate toplevel widget, and put them back in the container later. For example, you could compare two graphs side-by-side by tearing them out, and then replace them when you are finished.

A container may contain an unlimited number of folders. If there are too many tabs to view, you can arrange them as multiple tiers or scroll the tabs. The container uses the conventional Tk scrollbar syntax, so you can attach a scrollbar too.

EXAMPLE

You create a container widget with the **container** command.

```
# Create a new container
container .c
```

A new Tcl command **.c** is also created. This command can be used to query and modify the container. For example, to change the default borderwidth, you use the new command and the container's **configure** operation.

```
# Change the default font.
.c configure -borderwidth 2
```

You can then add folders using the **insert** operation.

```
# Create a new folder "f1"
.c coinsert 0 "f1"
```

This inserts the new tab named "f1" into the container. The index **0** indicates location to insert the new tab. You can also use the index **end** to append a tab to the end of the container. By default, the text of the tab is the name of the tab. You can change this by configuring the **−text** option.

```
# Change the label of "f1"
.ts tab configure "f1" -label "Tab #1"
```

The **insert** operation lets you add one or more folders at a time.

```
.ts insert end "f2" -label "Tab #2" "f3" "f4"
```

The tab on each folder contains a label. A label may display both an image and a text string. You can reconfigure the tab's attributes (foreground/background colors, font, rotation, etc) using the **tab configure** operation.

```
# Add an image to the label of "f1"
set image [image create photo -file stopsign.gif]
.ts tab configure "f1" -image $image
.ts tab configure "f2" -rotate 90
```

Each folder may contain an embedded widget to represent its contents. The widget to be embedded must be a child of the container widget. Using the **−window** option, you specify the name of widget to be embedded. But don't pack the widget, the container takes care of placing and arranging the widget for you.

```
graph .ts.graph
.ts tab configure "f1" -window ".ts.graph" \
    -fill both -padx 0.25i -pady 0.25i
```

The size of the folder is determined the sizes of the Tk widgets embedded inside each folder. The folder will be as wide as the widest widget in any folder. The tallest determines the height. You can use the tab's **−pagewidth** and **−pageheight** options override this.

Other options control how the widget appears in the folder. The**−fill** option says that you wish to ha ve the widget stretch to fill the available space in the folder.

```
.ts tab configure "f1" -fill both -padx 0.25i -pady 0.25i
```

Now when you click the left mouse button on "f1", the graph will be displayed in the folder. It will be automatically hidden when another folder is selected. If you click on the right mouse button, the embedded widget will be moved into a toplevel widget of its own. Clicking again on the right mouse button puts it back into the folder.

If you want to share a page between two different folders, the **−command** option lets you specify a Tcl command to be invoked whenever the folder is selected. You can reset the **−window** option for the tab whenever it's clicked.

```
.ts tab configure "f2" -command {
    .ts tab configure "f2" -window ".ts.graph"
}
.ts tab configure "f1" -command {
    .ts tab configure "f1" -window ".ts.graph"
}
```

If you have many folders, you may wish to stack tabs in multiple tiers. The container's **−tiers** option requests a maximum number of tiers. The default is one tier.

```
.ts configure -tiers 2
```

If the tabs can fit in less tiers, the widget will use that many. Whenever there are more tabs than can be displayed in the maximum number of tiers, the container will automatically let you scroll the tabs. You can even attach a scrollbar to the container.

```
.ts configure -scrollcommand { .sbar set }  -scrollincrement 20
.sbar configure -orient horizontal -command { .ts view }
```

By default tabs are along the top of the container from left to right. But tabs can be placed on any side of the container using the **−side** option.

```
# Arrange tabs along the right side of the container.
.ts configure -side right -rotate 270
```

SYNTAX

The **container** command creates a new window using the *pathName* argument and makes it into a container widget.

container *pathName* ?*option value*?...

Additional options may be specified on the command line or in the option database to configure aspects of the container such as its colors, font, text, and relief. The **container** command returns its *pathName* argument. At the time this command is invoked, there must not exist a window named *pathName*, but *pathName*'s parent must exist.

When first created, a new container contains no tabs. Tabs are added or deleted using widget operations described below. It is not necessary for all the tabs to be displayed in the container window at once; commands described below may be used to change the view in the window. Containers allow scrolling of tabs using the **−scrollcommand** option. They also support scanning (see the **scan** operation). Tabs may be arranged along any side of the container window using the **−side** option.

The size of the container window is determined the number of tiers of tabs and the sizes of the Tk widgets embedded inside each folder. The widest widget determines the width of the folder. The tallest determines the height. If no folders contain an embedded widget, the size is determined solely by the size of the tabs.

You can override either dimension with the container's **−width** and **−height** options.

CONTAINER OPERATIONS

All **container** operations are invoked by specifying the widget's pathname, the operation, and any arguments that pertain to that operation. The general form is:

pathName operation ?*arg arg ...*?

Operation and the *arg*s determine the exact behavior of the command. The following operations are available for container widgets:

pathName **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **configure** operation described below.

pathName **configure** ?*option*? ?*value option value ...*?

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* and *value* are described below:

−background *color*
Sets the border color of the container.

−borderwidth *pixels*
Sets the width of the 3−D border around the outside edge of the widget. The **−relief** option determines how the border is to be drawn. The default is **2**.

−command *pattern*
Specifies to search for a window whose **WM_COMMAND** property matches the given pattern (X11 only). If no windows, or more than one window, matches the pattern, an error is generated. If *pattern* is the empty string, then no command search is performed. The default is "".

−cursor *cursor*
Specifies the widget's cursor. The default cursor is "".

−height *pixels*
Specifies the requested height of widget. If *pixels* is 0, then the height is height the embedded window plus the specified borderwidth. The default is **0**.

−highlightbackground *color*
Sets the color to display in the traversal highlight region when the container does not have the input focus.

−highlightcolor *color*
Sets the color to use for the traversal highlight rectangle that is drawn around the widget when it has the input focus. The default is **black**.

−highlightthickness *pixels*
Sets the width of the highlight rectangle to draw around the outside of the widget when it has the input focus. *Pixels* is a non-negative value and may have any of the forms acceptable to **Tk_GetPixels**. If the value is zero, no focus highlight is drawn around the widget. The default is **2**.

−name *pattern*
Specifies to search for a window whose **WM_NAME** property matches the given pattern (X11 only). If no windows, or more than one window, matches the pattern, an error is generated. If*pattern* is the empty string, then no name search is performed. The def ault is "".

−relief *relief*
Specifies the 3-D effect for the container widget. *Relief* specifies ho w the container should appear relative to widget that it is packed into; for example, **raised** means the container should appear to protrude. The default is **sunken**.

−takefocus *focus*
Provides information used when moving the focus from window to window via keyboard traversal (e.g., Tab and Shift-Tab). If*focus* is **0**, this means that this windo w should be skipped entirely during keyboard traversal. **1** means that the this window should always receive the input focus. An empty value means that the traversal scripts decide whether to focus on the window. The default is **1**.

−width *pixels*
Specifies the requested width of the widget. If *pixels* is 0, then the width is the width the embedded window and the specified borderwidth. The default is **0**.

−window *id*
Specifies the foreign embedded using its path or X window id.

pathName **find** **−command**|**−name** *pattern*

Searches for all windows that match the given pattern. If the **−command** switch is given, all windows whose `WWM_COMMAND` property match *pattern* are returned in a list (X11 only). If the **−name** switch is given, all windows whose `WWM_NAME` property match *pattern* are returned in a list. The list returned will contains pairs of the window id and the matching property.

KEYWORDS

container, widget