width	*The new width* *Allowed: Integer*
whichbox	*which listbox gets the new width* *Allowed: An optional integer*

Description

This is an event which sets the width of one or all listboxes. If there is only one argument, all listboxes are set to that width. If there is a second argument, it specifies which listbox should be set to that width.

1.4.37 tabdialog - Tool

Package display
Module widgets

Tk widget where the menu label changes with selection
include "tabdialog.g"

Constructors

tabdialog	*Construct an tabdialog*

Functions

add	*Add a new Tab*
available	*Is this Tab available*
delete	*Delete an existing Tab*
deleteall	*Delete all of the Tabs*
dialogframe	*Get the internal dialog frame*
done	*Destroy ourselves*
front	*Bring this Tab to the front*
list	*List Tabs*
replace	*Replace a new Tab*
which	*Which Tab is visible*

Description

Glish/Tk does not yet come with a Tab widget. When we are bound to a richer widget set such as Tix, such a widget will be available.
In the mean time, this widget simulates a Tab widget with a button bar.
The widget provides you a frame into which you place your Tabs. For each Tab

there is a button; pushing it maps the Tab to the foreground and the previous one to the background. The active tab is denoted by a sunken relief for the button. Otherwise, buttons are shown with raised relief.

tabdialog - Constructor

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Construct an tabdialog

Synopsis
tabdialog(parent, colmax, title, hlthickness, widgetset)

Arguments

parent	*Parent of the tabdialog* *Allowed: Agent*
colmax	*Maximum number of Tab buttons per row* *Allowed: Integer* *Default: 5*
title	*Title place above button row* *Allowed: String* *Default: 'Select'*
hlthickness	*Thickness of tab border* *Allowed: Integer* *Default: 5*
widgetset	*The widget server to use* *Allowed: widgetserver tool* *Default: dws*

Returns
Agent

Description

This is the constructor. If you assign `title=unset` *then no title will appear. As many rows of buttons as needed will be created as you add TABs.*

Example

```
include 'tabdialog.g'
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1', 'Press for tab1')
t.add(frames['2'], 'TAB2', 'Press for yet another tab')
```

This will make one row of buttons with two Tabs.

---

tabdialog.add - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Add a new Tab

Synopsis
add(tab, tabname, hlp)

Arguments

tab	*The Tab* *Allowed: Agent*
tabname	*Name of the tab for the tab button* *Allowed: String*
hlp	*Popup help text for tab button* *Allowed: String*

Returns
Bool

Description

This function adds the specified Tab and adds a button with the given name. Note that adding a Tab does not make it the visible one unless it's the first Tab.

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
```

This will make one row of buttons with two Tabs. The first Tab will be the visible one.

---

tabdialog.available - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Is this Tab available

Synopsis

available(tabname)

Arguments

tabname	*Name of the tab for the tab button* *Allowed: String*

Returns
Bool

Description

This function indicates whether the specified Tab name has been added or not.

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
#
t.available('TAB1')
T
t.available('TAB10')
F
```

---

tabdialog.delete - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Delete an existing Tab

Synopsis
delete(tabname)

Arguments

tabname	*Name of the tab for the tab button* *Allowed: String*

Returns

Bool

Description

This function deletes the specified Tab. Subsequent additions will first fill this empty location.

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
t.delete('TAB1')
```

This will make one row of buttons with two Tabs. The first Tab will be the visible one. The first tab is then deleted.

---

tabdialog.deleteall - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Delete all of the Tabs

Synopsis
deleteall()

Returns
Bool

Description

This function deletes all of the Tabs.

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
#
t.deleteall()
```

---

tabdialog.dialogframe - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Get the internal dialog frame

Synopsis
dialogframe()

Returns
Agent

Description

This function recovers the internal Tab dialog frame into which you should place your Tabs.

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
```

---

tabdialog.done - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This function destroys the widget.

Example

```
- include 'tabdialog.g'
- f := dws.frame();
- t := tabdialog(f)
- t.done()
```

---

tabdialog.front - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Bring this Tab to the front

Synopsis

front(tabname)

Arguments

tabname	*Name of the tab* *Allowed: String*

Returns

Bool

Description

This function brings the specified Tab to the front.

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
#
t.front('TAB2')
```

---

tabdialog.list - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

List Tabs

Synopsis
list()

Returns
Vector of strings

Description

This function returns the list of Tabs.

Example

```
f := dws.frame();
t := tabdialog(f)
```

```
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
#
t.list()
TAB1 TAB2
```

---

tabdialog.replace - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Replace a new Tab

Synopsis
replace(tab, tabname)

Arguments

tab	*The Tab* *Allowed: Agent*
tabname	*Name of the tab for the tab button* *Allowed: String*

Returns
Bool

Description

This function replaces the specified pre-existing Tab.

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
#
frames['3'] := dws.frame(df)
frames['3']->unmap()
frames['3'].b := dws.button(frames['3'], 'b3')
#
t.replace(frames['3'], 'TAB2')
```

---

tabdialog.which - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Which Tab is visible

Synopsis
which()

Returns
Record

Description

This function returns a record containing the name of the Tab which is currently visible and its index (1 relative).

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
#
t.front('TAB2')
t.which()
[name=TAB2, index=2]
```

1.4.38 tabdialogtest - Function

Package display
Module widgets

Test tabdialog widget

Synopsis
tabdialogtest()

Returns
Agent

Description

This function tests the tabdialog widget. It makes a GUI with some buttons you can press.