Chapter 1

Display

package for image and data display

Description
The display package contains modules that support the display of AIPS++ Images, *Glish* arrays and other AIPS++ data.

1.1 guiutils - Module

GUI Related Utilities

Description
guiutils contain several graphic user interface (GUI) releated functions that allow a programmer/user to layer a GUI on top of an application. Please look at the "Writing a GUI for Your Application" in Note 197: How to Write an AIPS++Application.

Tools

barchart	Record values and display as a barchart
displaylist	Maintain an editable list of items
evalwindow	A simple GUI text editor from which Glish code may be evaluated.
gui	Standard Input Forms generated from Glish records
guiframework	Standard framework for Glish based AIPS++Graphical User Interfaces (GUI's)
messagebox	A simple popup text window for displaying short to medium-size messages.
progress	Display visually the progress of some task
single_entry_dialog_box	A popup dialog box for entering one item of text.
status_line	A one-line text widget for displaying status information.
tracewindow	A text display widget for tracing Glish function execution

Functions

addpopuphelp	add popup help to a GUI
choice	Have a user select one of a number of choices
choicewindow	Display a question in a window with multiple choice answers
chooser	AIPS++ chooser subsequence
datachooser	Data Chooser
displaylisttest	Test/demonstration of the displaylist class
donechoice	A choice dialog appropriate for done()
exitchoice	A choice dialog appropriate for exit
filechooser	File Chooser
infowindow	Display an informational message in a window
popuphelp	add popup help to a GUI
popupmenu	add a popup type selection help menu to a GUI
popupremove	remove popup help from a GUI
tablechooser	Table Chooser

1.1.1 choice - Function
Package display
Module guiutils

Have a user select one of a number of choices

Synopsis
choice(description, choices, types, default, interactive, timeout, widgetset)
include "choice.g"

Description

Choice is used to have the user select one of several strings. It puts up a gui box with some descriptive text, and a button for each choice. A default choice can be indicated. When left unset, the the first choice is the default. The default is used if it is not possible to put up a GUI or a timeout occurs.
This function is used by the tasking system, but might be of use to people writing their own scripts.
The types, when specified, should be chosen from those known to the widgetserver. These include "plain", "action", "halt", and "dismiss". When types is unset (its default value) all buttons will be "plain".
When a default is specified, it indicates which element of choices is returned when no GUI can be displayed or a timeout occurs.

Arguments

description	Descriptive text for the choices Allowed:string
choices	One of these choices will be returned Allowed:string array
types	Button types for choices Allowed:string array Default: unset
default	Which choice is the default one Allowed:integer Default: 1
interactive	If interactive, use a GUI, otherwise return the first choice. Allowed:boolean
timeout	How long to wait for the user to respond Allowed:integer
widgetset	Which widgetserver to use Allowed:widgetserver Default: dws

Returns
string

Example

```
# all plain buttons, default choice is no
result := choice('Do you want to quit?', ['no', 'yes'])
if (result == 'yes') {print 'Bye!'; exit; };
```

Example

```
# halt and dismiss button types, choice is yes
result := choice('Do you want to quit?', ['no', 'yes'],
                 ['dismiss', 'halt'], 2)
if (result == 'yes') {print 'Bye!'; exit;};
```

1.1.2 donechoice - Function

Package display
Module guiutils

A choice dialog appropriate for done()

Synopsis
donechoice(description, widgetset)
include "choice.g"

Description
This is a choice appropriate for use with a tool done function. Done which may be invoked by pressing a “Done” button on the tool GUI. Since done is used to destroy the tool, it is often desirable to ensure that the user really meant to invoke that function. This function exists so that the choice dialog boxes used with done all have a similar appearance. Only the descriptive text will vary. This function returns a boolean T if the user has pressed the "yes" button. In all other cases, a F will be returned.

Arguments

description	Descriptive text for the choices Allowed:string
widgetset	Which widgetserver to use Allowed:widgetserver Default: dws

Returns
boolean

Example

```
amidone := donechoice();
if (amidone) {print 'This tool has been destroyed';};
```

1.1.3 exitchoice - Function
Package display
Module guiutils

A choice dialog appropriate for exit

Synopsis
exitchoice(widgetset)
include "choice.g"

Description
This is a choice appropriate for use with a GUI “Exit” button. According to the GUI Guidelines, “Exit” is used to allow the user to exit from glish. It is usually desirable to ensure that the user really meant to exit glish. This function exists so that the choice dialog boxes used with “Exit” all have the same appearance.
This function returns a boolean T if the user has pressed the "yes" button. In all other cases, a F will be returned.

Arguments

widgetset	Which widgetserver to use Allowed:widgetserver Default: dws

Returns
boolean

Example

```
reallyexit := exitchoice();
if (reallyexit) {print 'Bye!'; exit;};
```

1.1.4 filechooser - Function
Package display
Module guiutils

File Chooser

Synopsis
filechooser(dirName, restrictions, title, wait, writeOK, multi, access)

Description

Provides AIPS++ with a standard file chooser. Typical use would be in one of three ways (there are lots of permuation).

filechooser() blocks, waits around for the user to make a choice and returns the result in a glish record, guiReturns='filename'. This will likely change so that it returns only the file name.

filechooser(wait=F) non-blocking, returns an agent. The agent signal a "returns" event when the OK button is pressed.

filechooser(multi=T) blocks, allows multiple choices. Returns the choices in a string vector.

You may use the restrictions argument to fine tune how the filechooser presents the directories and files.

restrictions =$[data = F, tables = F]$ file chooser default. Allows you to pick just files.

restrictions =$[data = T, tables = T]$ data chooser default. Allows you to pick tables and files and allows the user to choose what kinds of data to be able to pick.

restrictions =$[data = F, tables = T]$ table chooser default. Allows only tables to be choosen. No refinements of which tables to allowed.

restrictions =$[data = T, tables = F]$ not useful.

You may use the access argument to select only those files with certain access permissions. The access argument can contain any combination of the values "r", "w", and"x".

access ="r" the default value for all choosers. Only files with 'read' permission are displayed.

access ="w". Only files with 'write' permission are displayed.

access ="x". Only files with 'execute' permission are displayed.

access ="rw". Files which are both readable and writable are displayed. Other combinations are possible.

Arguments

dirName	Directory name Allowed:Any directory name Default: Current directory
restrictions	Whether to show tables or files Allowed:Two element boolean record Default: [T,T]
title	Title to put on Window Allowed:Any String Default: AIPS++ chooser
wait	Flag for whether to wait for result or not Allowed:T or F Default: T
writeOK	Flag for allowing writing of files Allowed:T or F Default: T
multi	Flag for allowing multiple choices Allowed:T or F Default: F
access	String specifying required access permissions of files Allowed:any combination of r,w, and x Default: r

Returns
An agent or record containing the file name

1.1.5 datachooser - Function
Package display
Module guiutils

Data Chooser

Synopsis
datachooser(dirName, title, wait, writeOK, multi, access)

Description
The data chooser allows the user to select aips++ tables and files. There are options available for limiting the scope of the data to be selected. You may choose to not block by setting wait=F.

Arguments

dirName	Directory name Allowed:Any directory name Default: Current directory
title	Title to put on Window Allowed:Any String Default: AIPS++ chooser
wait	Flag for whether to wait for result or not Allowed:T or F Default: T
writeOK	Flag for allowing writing of files Allowed:T or F Default: T
multi	Flag for allowing multiple choices Allowed:T or F Default: F
access	String specifying required access permissions of files Allowed:any combination of r,w, and x Default: r

Returns
An agent or record containing the table/file name

1.1.6 tablechooser - Function

Package display
Module guiutils

Table Chooser

Synopsis
tablechooser(dirName, title, wait, writeOK, multi, access)

Description
The table chooser allows the user to select only aips++ tables. You may choose to not block by setting wait=F.

Arguments

dirName	Directory name Allowed:Any directory name Default: Current directory
title	Title to put on Window Allowed:Any String Default: AIPS++ chooser
wait	Flag for whether to wait for result or not Allowed:T or F Default: T
writeOK	Flag for allowing writing of files Allowed:T or F Default: T
multi	Flag for allowing multiple choices Allowed:T or F Default: F
access	String specifying required access permissions of files Allowed:any combination of r,w, and x Default: r

Returns
An agent or record containing the table name

1.1.7 chooser - Function
Package display
Module guiutils

AIPS++ chooser subsequence

Synopsis
chooser(dirName, restrictions, title, wait, writeOK, multi, access)

Description
The chooser subsequence provides the actual gui for the file, data, and table choosers. You may call it directly but the preference is for glish programmers to use the file, data, or table chooser.

Arguments

dirName	Directory name Allowed:Any directory name Default: Current directory
restrictions	Whether to show tables or files Allowed:Two element boolean record Default: [T,T]
title	Title to put on Window Allowed:Any String Default: AIPS++ chooser
wait	Flag for whether to wait for result or not Allowed:T or F Default: T
writeOK	Flag for allowing writing of files Allowed:T or F Default: T
multi	Flag for allowing multiple choices Allowed:T or F Default: F
access	String specifying required access permissions of files Allowed:any combination of r,w, and x Default: r

Returns
An agent, sends a returns event when OK button is pressed

1.1.8 guiframework - Tool

Package display
Module guiutils

Standard framework for Glish based AIPS++Graphical User Interfaces (GUI's) include "guiframework.g"

Description
The guiframework (gf) provides a framework for constructing consistent graphical user interfaces. It handles the overhead of the standard set of widgets that most GUI's should have, menubars, command buttons, status lines. Customization of the menu bar is fairly straight forward with the use of Glish records.
A menu record consists of

```
menu_name::text              #! Text appearing on the Menu Button
menu_name.menu_item
```

```
menu_name.menu_item.text      #! Text appearing on the Menu Item
menu_name.menu_item.relief
menu_name.menu_item.action
menu_name.menu_item.type
menu_name.menu_item.state
```

An action record consists of

```
action_name.text
action_name.action
```

The functions, addactions and addmenus, expect the action and menu records to live off a root menu or action tree (i.e. menu_root.menu_name.menu_item). Any valid glishTk button attribute maybe set for each menu or action item. The following member functions illustrate how to construct the menu/action records.

default_filemenu make a simple menu,

default_options make a menus with cascading menus

defaultmenus put together a menus record

defaulthelp put together a help record

default_actions create an actions record

```
self.default_filemenu := function()
{  wider public;
   file := [=];
   file::text := 'File';
   file.open := [=];
   file.open.text := 'Open...';
   file.open.relief := 'flat';
   file.open.action := function(){
                return filechooser(title='AIPS++ Filechooser -- Open File');}
   file.save := [=];
   file.save.text := 'Save...'
   file.save.relief := 'flat';
   file.save.action := function(){
                return filechooser(title='AIPS++ Filechooser -- Save File');}
   file.blank := [=];
   file.blank.text := '';
   file.blank.relief :='flat';
   file.blank.disable :=T;

   file.bug := [=];
   file.bug.text :='Report Bug...';
```

```
   file.bug.relief :='flat';
   file.bug.action := bug;

   file.blank1 := [=];
   file.blank1.text := '';
   file.blank1.relief :='flat';
   file.blank1.disable :=T;

   file.close := [=];
   file.close.text := 'Close';
   file.close.relief := 'flat';
   file.close.action := public.dismiss;

   return ref file ;
}

self.default_options := function()
{
   options := [=];
   options::text := 'Options'
   options.browser := [=];
   options.browser::text := 'Browser'
   options.browser.text := 'Browser'
   options.browser.type := 'menu'
   options.browser.relief := 'flat'
#
   options.browser.menu := [=];
   options.browser.menu.netscape := [=];
   options.browser.menu.netscape.text := 'Netscape';
   options.browser.menu.netscape.type := 'radio';
   options.browser.menu.netscape.relief := 'flat';
   options.browser.menu.netscape.action := function(){ global helpsystem;
                                         helpsystem::browser := 'netscape';}
#
   options.browser.menu.mosaic := [=];
   options.browser.menu.mosaic.text := 'Mosaic';
   options.browser.menu.mosaic.type := 'radio';
   options.browser.menu.mosaic.relief := 'flat';
   options.browser.menu.mosaic.action := function(){global helpsystem;
                                         helpsystem::browser := 'mosaic';}
#
   options.browser.menu.other := [=];
   options.browser.menu.other.text := 'Other';
   options.browser.menu.other.type := 'radio';
   options.browser.menu.other.relief := 'flat';
   options.browser.menu.other.action := function(){ global helpsystem;
```

```
                                         helpsystem::browser := 'other';}
   return ref options;
}

   # Returns a reference to the default menus, file and options

public.defaultmenus := function()
{  wider self;
   menus := [=];
   menus.file := self.default_filemenu();
   menus.options := self.default_options();
   return ref menus;
}

   # Returns a reference to the default help menu

public.defaulthelp := function()
{
   hmenu := [=];
   hmenu::text := 'Help';
   hmenu.about := [=];
   hmenu.about.text := 'About';
   hmenu.about.relief := 'flat';
   hmenu.about.action := about;
#
   hmenu.refman := [=];
   hmenu.refman.text := 'Reference Manual';
   hmenu.refman.relief := 'flat';
   hmenu.refman.action := function()
                         { help('Refman:'); }
#
   hmenu.gstart := [=];
   hmenu.gstart.text := 'Getting Started';
   hmenu.gstart.relief := 'flat';
   hmenu.gstart.action := function()
                          { help('gettingstarted:'); }

   return ref hmenu;
}

   # Returns a reference to the default actions

public.defaultactions := function()
{  wider public;
   action := [=];
   action.apply := [=];
```

```
   action.apply.text := 'Apply';
   action.apply.action := function(){print 'Apply Pressed'};
   action.reset := [=];
   action.reset.text := 'Reset';
   action.reset.action := function(){print 'Reset Pressed'};
   action.cancel := [=];
   action.cancel.text := 'Cancel';
   action.cancel.action := public.dismiss;
   return ref action;
}
```

Here are a couple of simple examples that illustrate the use of guiframework.
First we construct a simple message window with a dismiss button.

```
# Info dialog

infowindow := function( some_text='Your slogan here.',
                        title='AIPS++ Infomation Window')
{
       #  Create the default action button, Dismiss

    action := [=];
    action.dismiss.text := 'Dismiss';

       #  Create the guiframework, title sets the title of the frame,
       #   the first F, tells the constructor not to use the default menus
       #   the second F, tells the constructor not to use the default help menu
       #   action, says create the action buttons (there's only one)

    a := guiframework(title, F, F, action);

       # We need set the dismiss handler after we create the tool otherwise
       # it wouldn't know how to dismiss itself.

    a.addactionhandler('dismiss', a.dismiss);

       # Now get the frame that will be the parent of all our application
       # gui components

    wf := a.getworkframe();

       # Attach a message to the work frame

    m := message(wf, some_text, width=600);
```

```
       # Return a reference to the info window.
    return ref a;
}
```

Here's a more sophisticated example. It implements a choices window.

```
choicewin := function(question, choices, interactive=have_gui(), timeout=150)
{
    if (!is_string(question) || !is_string(choices)) {
        fail '::choice(description,choices) : description and choices must be strings'
    }

    if (!interactive || !have_gui()) {
        return choices[1]
    }

       #  Setup up the buttons, the default action procedure for each button
       #  is to return its value.

    action := [=];
    for(choice in choices){
       action[choice] := [=];
       action[choice].text := choice;
    }
       # Create the guiframework, no menus, but as many actions as buttons

    a := guiframework("AIPS++ Please make a choice", F, F, action);
    wf := a.getworkframe();

       # Display the query in the workframe

    m := message(wf, question, width=600);

       # Now we setup the timing loop and do a blocking wait for a response

    done := F;
    timer := client("timer", 0.2);
    choiceIs := choices[1];

       # Put the amount of time left to choose in the status loop

    a.updatestatus(spaste('Time left to choose: ', timeout/5, ' s'));

       #  Wait for an answer or until we timeout
    while(!done){
         # Check if a handler has returned
       if(a.handle::returned){
```

```
            # Yup set the choice a break the loop
          choiceIs := a.handle::value;
          break;
       }
          # Nope, wait for the timer
       await timer->ready
       if ($name == "ready") {
  timeout -:= 1;
  done := timeout <= 0;
          timestring := '     ';
          if(!(timeout%5)){
                # Update the status line with the timeleft, count down seconds
             timestring := spaste('Time left to chose: ', timeout/5, ' s');
     a.updatestatus( timestring);
          }
        }
    }

      # All done, cleanup timer and window and return the choice

    send timer->terminate();
    a.dismiss();
    return choiceIs;
}
```

Constructors

guiframework	Constructs the GUI framework, menus, work area, etc...

Functions

addactionhandler	Sets the handler for a menu or command buttion
addactions	Add buttons to command area.
addhelp	Add help menus to the menubar.
addmenus	Add menus to the menubar.
busy	Turns the icon into a watch or pointer. Disables the guiframework frame.
cleanup	removes agents generated by guiframework
defaultactions	Returns the default actions for command buttons.
defaulthelp	Returns the default help menu.
defaultmenus	Returns the default menus.
dismiss	Dismisses the main window.
getworkframe	Returns a reference to the frame where most GUI components are found.
updatestatus	Updates the status line on the window.
updatetitle	Updates the title line on the frame/window.

guiframework - Constructor

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Constructs the GUI framework, menus, work area, etc...

Synopsis

guiframework(title, menus, helpmenu, actions, guiframenote, parent)

Arguments

title	Window title Allowed:Any string Default: AIPS++Window
menus	menu records Allowed:Glish menus records Default: Default file and options menus
helpmenu	help record Allowed:Glish help menu record Default: Default help menus
actions	Action record Allowed:Glish action record Default: Default action buttons
guiframenote	note function Allowed:any note function Default: note
parent	Parent frame Allowed:Frame to attach guiframework Default: F

Description

The guiframework provides a consistent look and feel GUI for gui applications that use it. With it you can specify standard menus, help and actions.
If the parent frame is `F` guiframe work will generate the top-level frame.
Often we need to change the state of button, grey it, change a button label or the like. Rather than provide lots of guiframework functions to do these things, there is a backdoor using the app record. The app datamember of guiframework contains all the GUI elements defined by a guiframework. The scheme
The record `app.mb.btns` contain all the buttons in the menubar, a button is identified by the menu name and the menu item name. For example the 'file'

menu is identified `app.mb.btns.file`, the open option on file is `app.mb.btns.fileopen`, menu items concatinated the menu name and menu item name to identify themselves.
Action buttons belong to the `app.cmd.b` record. A go action would be `app.cmd.b.go`.

Returns
A standard AIPS++GUI framework window

---

guiframework.busy - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Turns the icon into a watch or pointer. Disables the guiframework frame.

Synopsis
busy(busyFlag)

Arguments

busyFlag	Busy Flag Allowed:T or F, T for busy, F for not busy Default: T

---

guiframework.dismiss - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Dismisses the main window.

Synopsis
dismiss()

Returns

Removes the window from the display

---

guiframework.defaultmenus - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Returns the default menus.

Synopsis
defaultmenus()

Returns
Returns a record of the default menus, file & options

---

guiframework.defaulthelp - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Returns the default help menu.

Synopsis
defaulthelp()

Returns
Returns a record of the default help menu.

---

guiframework.defaultactions - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Returns the default actions for command buttons.

Synopsis
defaultactions()

Returns
Returns a record of the default action buttions, Apply, Reset, and Cancel.

---

guiframework.addmenus - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Add menus to the menubar.

Synopsis
addmenus(menus)

Arguments

menus	Record, containing menus Allowed:Any Glish menu record

Returns
Nothing

---

guiframework.addhelp - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Add help menus to the menubar.

Synopsis
addhelp(helpmenu)

Arguments

helpmenu	Help menu record Allowed:Glish menu record

Returns
Nothing

---

guiframework.addactions - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Add buttons to command area.

Synopsis
addactions(actions)

Arguments

actions	Action buttons Allowed:Glish action record

Returns
Nothing

---

guiframework.getworkframe - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Returns a reference to the frame where most GUI components are found.

Synopsis
getworkframe()

Returns

A reference to frame which you use for the parent of GUI components.

---

guiframework.updatestatus - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Updates the status line on the window.

Synopsis
updatestatus(status)

Arguments

status	Status line Allowed:Any string

Returns
Nothing

---

guiframework.updatetitle - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Updates the title line on the frame/window.

Synopsis
updatetitle(status)

Arguments

status	Status line Allowed:Any string

Returns

Nothing

---

guiframework.addactionhandler - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Sets the handler for a menu or command buttion

Synopsis
addactionhandler(button, action)

Arguments

button	button name Allowed:String Default: Valid button name
action	action functions Allowed:Glish function Default: Valid function

Returns
Nothing

Description
Sometimes it's not possible to prescribe a button/menu action before construction of the framework. Addactionhandler allows you do this after the tool is created. A typical example is a menu exit or an action dismiss button. The following code excerpt illustrates adding a command action.

```
action := [=];
action.dismiss := [=];
action.dismiss.text = 'Dismiss';

a := guiframework('My window', F, F, action);
a.addactionhandler('dismiss', a.dismiss());
```

This code excerpt illustrates adding a menu action. Note that for menus we concatenate the menu button names to produce the proper button name. So, the dismiss option of the file menu has a name of filedismiss. If you have cascading menus, you just keep concatenating names, i.e. options, browser, netscape becomes optionsbrowsernetscape.

```
menus := [=];
menus.file := [=];
menus.file::text := 'File';
.
.
.
menus.file.dismiss := [=];
menus.file.dismiss.text := 'Dismiss'
menus.options := [=];
.
.
.
a := guiframework('My app', menus, T, T)
a.addactionhandler('filedismiss', a.dismiss);
```

---

guiframework.cleanup - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

removes agents generated by guiframework

Synopsis
cleanup()

Returns
Nothing

1.1.9 gui - Tool

Package	**display**
Module	**guiutils**

Standard Input Forms generated from Glish records
include "inputframe.g"

Description

The GUI tool provies a way for generating standard input forms. Input forms are supported in two ways:

1. Stand-alone input form,
2. Input form imbedded in a glishTk frame,

Additionally the input forms may be selected either via tabs or a scrolling list. The inputform and tabform methods use a *screen input record* or a collection of screen input records (passed as a record). The *screen input record* has the following members

title	text in the window title
label	text in the window
actions	(a.label a.function)record of functions
layout	ignored for now
data	record of datums
progress	T or F (Shows a progress bar, not implemented)

Each member of a *data record* is a datum and has the following members

label	Description of datum field
type	one of the following: string, vector_string, float, double long, integer, file, table, time, date, or text - string with more than oneline of text
required	boolean, T if required for processing
hint	check, radio, list, menu
enums	vector of allowed values
multiple	boolean for allowing multiple choices of enums
range.min	minimum value
range.max	maximum value
default	default value
verify	function to verify input
mask	display mask (not implemented)
help.url	a URL to drive a web browser (not implemented)
help.text	text for help of this field (not implemented)

If a user keys in a value starting with a $, the Glish variable with that same name (minus the $) will be used for that input value.

Constructors

Functions

inputform	Creates a input form, either stand-alone or in a frame
tabform	Creates a tab-styled input form, either stand-alone or in a frame

gui.inputform - Function

Package	**display**
Module	**guiutils**
Tool	**gui**

Creates a input form, either stand-alone or in a frame

Synopsis

inputform(scrnInputs, title, dismiss, parent)

Arguments

scrnInputs	Screen input record Allowed:any valid screen input record
title	Label or window title of input form Allowed:Any String Default: AIPS++ Input Form
dismiss	Dismiss flag Allowed:Flag to dismiss the form after returning from one of the action events Default: T
parent	The parent frame Allowed:Either F or the frame to put the input form Default: F

---

gui.tabform - Function

Package	**display**
Module	**guiutils**
Tool	**gui**

Creates a tab-styled input form, either stand-alone or in a frame

Synopsis

tabform(methods, title, dismiss, parent, tabcount, side)

Arguments

methods	Collection of screen input records used by inputform Allowed: any valid record of screen input records	
title	Label or window title of input form Allowed: Any String Default: AIPS++ Tab Input Form	
dismiss	Dismiss flag Allowed: Flag to dismiss the form after returning from one of the action events Default: T	
parent	The parent frame Allowed: Either F or the frame to put the tab-input form Default: F	
tabcount	Number of functions that cause a list box to be used Allowed: any integer but should be less than 7 Default: 5	
side	Which side to pack the list box on Allowed: top, bottom, left, right Default: left	

Description
The function's argument may have a two attributes, disallow and show. The disallow attributes tells the tabframe not to allow the array of functions to display or execute that function. The show option will gray out a function in the tab list rather than hide it.

Several utility functions are available via the guiutils module. include "guicomponents.g"

Description

1.1.10 status_line - Tool

Package display
Module guiutils

A one-line text widget for displaying status information.
include "guicomponents.g"

Description
Many GUI programs will be easier to use if status information is displayed at

appropriate times for the user to read. This very simple widget provides a convenient and standard way to do this.

Constructors

status_line	Construct a status line widget.

Functions

append	Add text to the end of the current text.
clear	Remove all text from the widget.
delete	Destroy the status_line widget
show	Clear the line, and display this message

status_line - Constructor

Package	**display**
Module	**guiutils**
Tool	**status_line**

Construct a status line widget.

Synopsis
status_line(parentFrame)

Arguments

parentFrame	a glishtk frame widget

Returns
a status_line tool

---

status_line.show - Function

Package	**display**
Module	**guiutils**
Tool	**status_line**

Clear the line, and display this message

Synopsis
show(message)

Arguments

message	a new status message Allowed:any string

Returns
T

Description

---

status_line.clear - Function

Package	**display**
Module	**guiutils**
Tool	**status_line**

Remove all text from the widget.

Synopsis
clear()

Returns
T

---

status_line.append - Function

Package	**display**
Module	**guiutils**
Tool	**status_line**

Add text to the end of the current text.

Synopsis
append(message)

Arguments

message	Allowed:any string

Returns
T

Description
A blank space is prepended to [message], and the result is written to the end of the current text displayed in the widget.

---

status_line.delete - Function

Package	**display**
Module	**guiutils**
Tool	**status_line**

Destroy the status_line widget

Synopsis
delete()

Returns
T

1.1.11 messagebox - Tool

Package	**display**
Module	**guiutils**

A simple popup text window for displaying short to medium-size messages. include ”guicomponents.g”

Description
This widget creates its own toplevel frame, displays the supplied message, and

deletes itself when the delete member function is called, or the user presses the "Dismiss" button.

Constructors

messagebox	Create a message box.

Functions

delete	Destroy the messagebox window.
text	Erase any existing text, and display the new message.

messagebox - Constructor

Package	**display**
Module	**guiutils**
Tool	**messagebox**

Create a message box.

Synopsis
messagebox(message, background)

Arguments

message	The text to display Allowed:any string
background	background color Allowed:any color available on your system. Default: white

Returns
A messagebox tool

Description
Newlines can be embedded in the message. The window has quite a wide default width, about two-thirds the width of the average display screen. If the longest line of your message is shorter than that, the messagebox widget shrinks in width to be just the size of the longest line. No scrollbars are provided to accomodate wider or longer messages.

---

messagebox.text - Function

Package	**display**
Module	**guiutils**
Tool	**messagebox**

Erase any existing text, and display the new message.

Synopsis
text(message)

Arguments

message	New text to display Allowed:any String, newlines included

Returns
T

---

messagebox.delete - Function

Package	**display**
Module	**guiutils**
Tool	**messagebox**

Destroy the messagebox window.

Synopsis
delete()

Returns
T

1.1.12 single_entry_dialog_box - Tool

Package	**display**
Module	**guiutils**

A popup dialog box for entering one item of text.
include "guicomponents.g"

Description
This popup widget provides a way to get input from the user via a transient window. The programmer specifies a prompt, and an optional one-argument callback function – which will be called with the entered text as its sole argument. When the user types [return] in the entry widget, or presses the 'Okay' button, the popup window is destroyed, and the callback function is called. If the user presses the 'Dismiss' button, then the popup window is destroyed, and the callback function is not called.

Constructors

single_entry_dialog_box	Create the popup.

Functions

delete	Destroy the single_entry_dialog_box.

single_entry_dialog_box - Constructor

Package	**display**
Module	**guiutils**
Tool	**single_entry_dialog_box**

Create the popup.

Synopsis
single_entry_dialog_box(prompt, callbackFunction)

Arguments

prompt	Text to display, as a label, to the left of the text entry widget Allowed:any string Default: "
callbackFunction	Called at normal termination Allowed:Any one-argument Glish function. Default: F

Returns
A single_entry_dialog_box

Description
There are occasions in any GUI program where you need the user to enter a small amount of text. Sometimes the best way to get that text is with a popup window containing a text entry widget, and some appropriate termination buttons. A good way to make this operation modeless (i.e., run asynchronously) is to provide a function which will be called when the dialog box terminates; this widget returns the user's text as the first argument to the callback function.

Example

```
callback := function (stringValue) {
   print 'the user is: ', stringValue;
   }
nameDialog := single_entry_dialog_box ('Your name', callback);
```

---

single_entry_dialog_box.delete - Function

Package	**display**
Module	**guiutils**
Tool	**single_entry_dialog_box**

Destroy the single_entry_dialog_box.

Synopsis
delete()

Returns
T

1.1.13 evalwindow - Tool

Package	**display**
Module	**guiutils**

A simple GUI text editor from which Glish code may be evaluated.

include "guicomponents.g"

Description
This top level window provides a text editor specialized for Glish. It is specialized in the sense that any text you type can be sent to the Glish interpreter for evaluation. This makes it a good environment for developing (at least) small amounts of Glish code. As of now (December 1996) there are many standard editor features which are not yet available: saving text to a file, reading from a file, inserting text, pasting text, searching text. But all of this is possible in principle because the underlying tk 'Text' widget is quite powerful. Once these capabilities are developed, and sensibly presented in this composite widget, the 'evalwindow', it may be a good environment for the development of even large amounts of Glish code.

See Also
AIPS++Note 194: The SDCalc User Environment

Constructors

evalwindow	Create a Glish-specific editor

Functions

clear	Deletes all the text in the window.
contents	Get all of the text in the window
delete	Destroy the evalwindow

evalwindow - Constructor

Package	**display**
Module	**guiutils**
Tool	**evalwindow**

Create a Glish-specific editor

Synopsis
evalwindow(title, columns, rows)

Arguments

title	A label for the window Allowed:any string Default: ”	
columns	Width in characters Allowed:any integer Default: 60	
rows	Height in lines Allowed:any integer Default: 20	

Returns
An evalwindow tool

---

evalwindow.contents - Function

Package	**display**
Module	**guiutils**
Tool	**evalwindow**

Get all of the text in the window

Synopsis
contents()

Returns
A string, including newlines

---

evalwindow.clear - Function

Package	**display**
Module	**guiutils**
Tool	**evalwindow**

Deletes all the text in the window.

Synopsis
clear()

Returns

T

---

evalwindow.delete - Function

Package	**display**
Module	**guiutils**
Tool	**evalwindow**

Destroy the evalwindow

Synopsis
delete()

Returns
T

1.1.14 tracewindow - Tool

Package	**display**
Module	**guiutils**

A text display widget for tracing Glish function execution
include "guicomponents.g"

Description
The tracewindow is a work in progress. As of this writing (December 1996) the best use of the tracewindow is for the simple display of messages from a running Glish application. (See sdavg.g for the first example of this.) As any application performs its work, some of its operations will be performed behind the scenes, hidden from the user's view; this is especially true of a GUI application. Some users will want to know what is going on behind the scenes, and the well-written program will display those operations as actual Glish function calls, by sending appropriate text to the tracewindow.
The next level of tracing is only hinted at right now (December 1996). The goal – partly accomplished with some global functions located at the bottom of guicomponents.g – is to provide a tool for the user to select any function that appears in the tracewindow, and to view the Glish code that makes up that function. This approach, if properly supported, would allow an interested user to navigate at will through Glish code, starting with the function calls the

application prints to the tracewindow, and going just as deep, or as far afield, as she wishes.
This window is primarily useful when used as part of a properly constructed Glish GUI appplication. Such an application will, at the user's request, print the text of crucial function calls and variables to the tracewindow, and will ensure that important lower level functions have text representations for display here in the tracewindow as well. (Please see view_function_test () and view_function_test::text in guicomponents.g.)

See Also
AIPS++Note 194: The SDCalc User Environment

Constructors

tracewindow	Create a window for navigating Glish code.

Functions

append	Display newString on a new line, at the bottom of existing text
clear	Delete all of the text in the window
delete	Destroy the tracewindow
view_named_function	Display the text of the function

tracewindow - Constructor

Package	**display**
Module	**guiutils**
Tool	**tracewindow**

Create a window for navigating Glish code.

Synopsis
tracewindow(title, columns, rows)

Arguments

title	A label for the window Allowed:any string Default: ”	
columns	Width in characters Allowed:any integer Default: 50	
rows	Height in lines Allowed:any integer Default: 12	

Returns
a tracewindow tool

---

tracewindow.append - Function

Package	**display**
Module	**guiutils**
Tool	**tracewindow**

Display newString on a new line, at the bottom of existing text

Synopsis
append(newString)

Arguments

newString	
	Allowed:any string

Returns
T

---

tracewindow.clear - Function

Package	**display**
Module	**guiutils**
Tool	**tracewindow**

Delete all of the text in the window

Synopsis
clear()

Returns
T

---

tracewindow.delete - Function

Package	**display**
Module	**guiutils**
Tool	**tracewindow**

Destroy the tracewindow

Synopsis
delete()

Returns
T

---

tracewindow.view_named_function - Function

Package	**display**
Module	**guiutils**
Tool	**tracewindow**

Display the text of the function

Synopsis
view_named_function(functionName)

Arguments

functionName

Returns
T

1.1.15 progress - Tool

Package	**display**
Module	**guiutils**

Display visually the progress of some task
include "progress.g"

Description

A progress bar is used to track the progress of some task. It has a "meter" with a sliding line that advances across the window as the task executes. The progress bar will normally disappear when the task has reached its maximum value. The progress bar can optionally estimate the time remaining in the task by assuming that the remaining portion will complete at the same rate as the portion completed to date.
If X windows is not available (`have_gui() == F`) then this tool just does no-op's.

Example

```
- include "progress.g"
- bar := progress(-100, 100, 'main title', 'subtitle')
- bar.update(-75)
- bar.update(-50)
- ...
- bar.update(100)      # GUI window is removed
```

progress - Constructor

Package	**display**
Module	**guiutils**
Tool	**progress**

Create a progress meter

Synopsis

progress(min, max, title, subtitle, minlabel, maxlabel, estimate_time, barwidth, barheight, startopen)

Description

Create a progress meter. If `have_gui()` is false (for example, no DISPLAY environment variable is set) this results in a no-op tool. You generally do not need to set `minlabel` and `maxlabel`, since the defaults are generally what you want.

Note that `min` may in fact be greater than `max`. If so, the progress meter will "count down" correctly.

The `startopen` parameter is a little unusual. If it is a boolean (tt T or `F`) then, if true, the progress meter starts with the GUI "on". If a number, the progress meter's GUI turns on after the supplied number of seconds has elapsed. The reason this has been implemented is that it is distracting to see very short-lived progress meters popping up and disappearing very quickly.

Arguments

min	minimum value Allowed:Numeric
max	maximum value Allowed:Numeric
title	window title Allowed:String Default: ""
subtitle	bar title Allowed:String Default: ""
minlabel	left label Allowed:String Default: "" (minimum value)
maxlabel	right label Allowed:String Default: "" (maximum value)
estimate_time	estimate time remaining? Allowed:Boolean Default: T
barwidth	width of bar in pixels Allowed:Integer Default: 200
barheight	height of bar in pixels Allowed:Integer Default: 20
startopen	start with the window displayed? Or, how long to wait before displaying? Allowed:Boolean or Integer Default: 5

Returns
progress tool

---

progress.update - Function

Package	**display**
Module	**guiutils**
Tool	**progress**

Update a progress meter

Synopsis
update(value, autodisable)

Description
Updates the position of the bar on the progress meter. If `autodisable` is true, it will remove the window if it reaches the maximum value.

Arguments

value	New value of the progress meter Allowed:Numeric
autodisable	Remove the window when "max" is reached? Allowed:Boolean Default: T

Returns
Real - the fraction of the bar that is complete, 0.0-1.0.

---

progress.activate - Function

Package	**display**
Module	**guiutils**
Tool	**progress**

Remove the window from the screen

Synopsis

activate()

Description
Remove the progress meter window from the screen. If the window is already inactive, the function is a no-op.

---

progress.deactivate - Function

Package	**display**
Module	**guiutils**
Tool	**progress**

Return the window to the screen

Synopsis
deactivate()

Description
If the window is not currently active on the screen, this function returns it, even if the window has previously hit its maximum. If the window is already active, this function is a no-op.

1.1.16 displaylist - Tool

Package	**display**
Module	**guiutils**

Maintain an editable list of items
include "displaylist.g"

Description

A display list is an editable list of items. An item may either be in the "draw" list, or it may be in the "clipboard". An item moves back and forth via `cut`, `copy`, and `paste` operations.
This class is not intended to be used by general users, rather it is intended to be used in the constructions of GUI's.

Example
See global function, `displaylisttest()`

displaylist - Constructor

Package	**display**
Module	**guiutils**
Tool	**displaylist**

Create a displaylist tool

Synopsis
displaylist(parentframe, formatfunction)

Description
Create a list tool ready to start adding values to. Unless a frame is given to the constructor, the default is to not have a GUI representation of the displaylist. Generally you should provide a function to turn inserted values into strings to display in the list. By default, `as_string()` will be used, which is rarely what you will want.

Arguments

parentframe	frame to use for gui Allowed:frame or F Default: F (no frame)
formatfunction	turn list values into string for display Allowed:function

Returns
displaylist tool

---

displaylist.add - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

add a value to the display list

Synopsis
add(value)

Description
This function adds a new value to the draw list. Note that you may add any type of value to the display list, although generally you will add record types

Arguments

value	item to add to the draw list Allowed:any

Returns
number of entries in the draw list

---

displaylist.get - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

get an item from the displaylist

Synopsis
get(num, fromdrawlist)

Description
Use an index (1 relative) to get an item from either the draw list or the clipboard (by default the draw list). You can determine how many items there are by calling `ndrawlist()` or `nclipboard()`.

Arguments

num	index Allowed:integer
fromdrawlist	get from drawlist? Allowed:T

Returns
the item or fail if index invalid

---

displaylist.set - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

change or add an item in displaylist

Synopsis
set(num, value)

Description
Change or an existing item in the draw list (only) or add a new one. An existing item is changed if the index is in the range 1–`ndrawlist()`, otherwise a new item is added. (The index should only be one larger than the current length of the drawlist).

Arguments

num	index Allowed:integer
value	item to change or add in the draw list Allowed:any

Returns
the new length of the draw list or fail if index invalid

---

displaylist.clear - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

clear the draw list and clipboard

Synopsis
clear()

Description
Empties all items in the draw list and clipboard.

---

displaylist.cleardrawlist - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

clear the draw list

Synopsis
cleardrawlist()

Description
Empties all items in the draw list.

---

displaylist.clearclipboard - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

clear the clipboard

Synopsis
clearclipboard()

Description
Empties all items in the clipboard.

---

displaylist.ndrawlist - Function

Package	**display**

Module **guiutils**
Tool **displaylist**

how many items are in the draw list

Synopsis
ndrawlist()

Description
Returns the number of items currently in the draw list.

Returns
integer

---

displaylist.nclipboard - Function

Package **display**
Module **guiutils**
Tool **displaylist**

how many items are in the clipboard

Synopsis
nclipboard()

Description
Returns the number of items currently in the clipboard.

Returns
integer

---

displaylist.cut - Function

Package **display**
Module **guiutils**
Tool **displaylist**

move items from the draw list to the clipboard

Synopsis
cut(drawnums)

Description
Cut is used to move items (desiganted as an array of integers) from the draw list to the clipboard.

Arguments

drawnums	index array Allowed:integer

Returns
the length of the drawlist or fail

---

displaylist.copy - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

copy items from the draw list to the clipboard

Synopsis
copy(drawnums)

Description
Copy is used to to copy items (desiganted as an array of integers) from the draw list to the clipboard.

Arguments

drawnums	index array Allowed:integer

Returns
the length of the drawlist or fail

---

displaylist.paste - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

copy items from clipboard to the drawlist

Synopsis
paste()

Description
Paste is used to copy items from the clipboard to the drawlist. At present all items from the clipboard are copied to the end of the drawlist. We should probably at least allow pasting into the middle of the draw list.

Returns
the length of the drawlist or fail

---

displaylist.compact - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

restore wasted space in tool

Synopsis
compact()

Description
After many editing operations there can be wasted space inside the tool. Compact reclaims this space. No visibile change occurs in the state of the tool.

---

displaylist.apply - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

call a function on items

Synopsis
apply(fun, dodraw)

Description
Apply applies a function to every item in the draw list (default) or clipboard. The function must take a single argument (or all the other arguments must default), *i.e.*, the item it is being passed.

Arguments

fun	the function we are applying to items Allowed:function
dodraw	apply to items in draw list or clipboard Allowed:boolean Default: T (draw)

Returns
the length of the drawlist or or clipboard or fail

---

displaylist.lastchange - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

return a number that specifies the state of the displaylist

Synopsis
lastchange()

Description

It is sometimes useful to know whether or not the displaylist has changed since the last time you have looked at it (given that the displaylist will often be under user control through some GUI). If the number returned by this function has changed since the last time you called it, it has. The number returned is always non-negative, so you can initialize your cache with a negative number.

Returns
integer

---

displaylist.gui - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

show the displaylist in a GUI

Synopsis
gui(parentframe)

Description
Create a GUI for the displaylist using a supplied frame. The clipboard appears in a listbox on the left, the draw list in a listbox on the right, and various buttons between them to control cutting, pasting, *etc.* Items are selected by clicking and dragging in the listboxes.

Arguments

parentframe	parent frame to write into Allowed:frame

Returns
T or fail

---

displaylist.nogui - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

turn off GUI view of displaylist

Synopsis
nogui()

Description
Turns off the GUI representation of this display list, but does not clear the actual items in the draw list and clipboard. Note that this function sets the parentframe of `gui()` to F.

---

displaylist.save - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

save the displaylist into a file

Synopsis
save(filename)

Description
This function saves the displaylist (including the clipboard), into a file. At present, the file format is just a Glish value, but this is subject to change. If the file already exists, it is overwritten. A fail is returned if the file cannot be written to (probably a read-only or non-existent directory).

Arguments

filename	file to write into Allowed:string

Returns
T or fail

---

displaylist.restore - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

restore the displaylist from a file

Synopsis
restore(filename)

Description
This function restores the displaylist from a file. The current contents are removed first. At present there is no way to merge from a file, but this could be added if desired.

Arguments

filename	file to read from Allowed:string

Returns
T or fail on read failure

---

displaylist.setselectcallback - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

call a function when an item is selected

Synopsis
setselectcallback(callback)

Description
This sets a callback function to be called whenever an item is selected in the draw list of the GUI. The function must take one argument - the selected item.

Arguments

callback	function to call on item Allowed:function

Returns
T or fail

---

displaylist.setchangecallback - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

call a function when the list changes

Synopsis
setchangecallback(callback)

Description
This sets a callback function to be called when the list is first changed, at which point the callback is unset. The idea is that the user somehow "saves" the drawlist, at which point he wants to be notified when the list has changed. When he has saved it again, he resets the callback.

Arguments

callback	function to call on item Allowed:function

Returns
T or fail

---

displaylist.setrefreshcallback - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

call function on cut and paste

Synopsis
setrefreshcallback(callback)

Description
Set a function to be called whenever there is a cut or paste. Arguably, the function should be called at other times (e.g., copy), or at least there should be additional callback functions. This function must take no arguments.

Arguments

callback	function to call on item Allowed:function

Returns
T or fail

---

displaylist.done - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

release/destory this tool

Synopsis
done()

Description
Destroy all resources (GUI and the item lists) allocated by this tool.

1.1.17 displaylisttest - Function

Package	**display**
Module	**guiutils**

Test/demonstration of the displaylist class

Synopsis
displaylisttest()

Description
Combination test/demonstration of displaylist.

1.1.18 popuphelp - Function

Package display
Module guiutils

add popup help to a GUI

Synopsis
popuphelp(fr, txt, hlp, combi, width)
include "popuphelp.g"

Description
`popuphelp` Is a function which adds the code necessary to add "popup" help to an agent. In particular, when your cursor "lingers" or "hovers" over that agent (e.g. a button) for more than about half a second, the help will popup, and go away when the cursor leaves the agent or when a button is pushed, etc.
Help text comes in two flavours: short and long. If only one of them is available, they will both be identical. Short help is displayed on one line if it is not longer than the specified width (60 by default), else on several lines. If no explicit long help is present, the shorthelp is truncated at a new line or at a period, and will never be longer than the specified width. If the longhelp differs from the shorthelp (either by truncation or otherwise), the shorthelp is extended with ".." at the end of the displayed short text.
Short help is displayed by default when the cursors hovers above the agent.
Long help is tied to the right mouse button by default.
Popup help can, dynamically, be tied by setting the 'system.help.type' to 0 (no popup), 1(short and long), 2(long only with MB3), 3(long only in hover).
A "popupmenu" function is also available to add a "?" button to your GUI which let the user choose the context popup type action. If a popupmenu is generated, the default can be set with the aipsrc variable "help.popup.type" (possible values: 'none', 'both', 'mb3long' and 'hoverlong'), or by the programmer.
For most popup help items you merely specify a text string that is the help text. For a listbox, you can supply a callback function that takes two arguments and returns a string. The two supplied arguments are:

1. the text of the listbox item the cursor is near; and

2. the Y-index (0 relative!) of the cursor position in the listbox.

It should be noted that timing errors could crash the system (or leave popuphelp floating around) if an agent that supports popuphelp is just zeroed (e.g. by frameagent :=F). A function 'popupremove(agent)' exists to do a proper zeroing.

Arguments

fr	agent to which popup text should be added Allowed:record or agent
txt	popup text, or, for listbox, possible callback routine Allowed:string or function as described above Default: F
hlp	optional separate short help text, or,for listbox, possible callback routine Allowed:string or function as described above Default: F
combi	specifies if short text should precede long text Allowed:bool Default: F
width	specifies maximum length of one shorthelp line Allowed:integer Default: 60

Returns
T (or fail)

Example

```
include 'popuphelp.g';
private := [=];
private.f := frame(height=100);
private.b := button(private.f);
popupmenu(private.f);  # to add a ? menu button (if wanted)
popuphelp(private.b, 'Press the button, and make this a long text');
private.l := listbox(private.f);
private.l->insert('AAA'); private.l->insert('BBB');
private.l.helpfct := function(name, index) {
    # Only need name here.
    return spaste('Your cursor is near ', name);
}
popuphelp(private.l, private.l.helpfct);
```

1.1.19 popupremove - Function

Package display
Module guiutils

remove popup help from a GUI

Synopsis
popupremove(ag, mxlevels)
include "popuphelp.g"

Description
`popupremove` Is a function which deletes any popuphelp (if present) from the specified agent, and sets the agent to False after having done it. Using it is necessary if, e.g., you want to zero the agent in a whenever: timing problems could crash the system if popuphelp is present and lingering somewhere.
The function works recursively, if fields in the argument are records.

Arguments

ag	agent which should be removed (possibly recursively) Allowed:agent Default: F
mxlevels	max. recursion level Allowed:Int Default: 8

Returns
T

Example

```
include 'popuphelp.g';
private := [=];
private.f := frame(height=100);
private.f.a := frame(private.f);
private.f.a.b := button(private.f.a);
popupmenu(private.f);  # to add a ? menu button (if wanted)
popuphelp(private.f.a.b, 'Press the button, and make this a long text');
private.l := listbox(private.f);
```

```
private.l->insert('AAA'); private.l->insert('BBB');
private.l.helpfct := function(name, index) {
    # Only need name here.
    return spaste('Your cursor is near ', name);
}
popuphelp(private.l, private.l.helpfct);
whenever private.f.a.b->press do {
    popupremove(private.f.a);
}
```

1.1.20 popupmenu - Function

Package display
Module guiutils

add a popup type selection help menu to a GUI

Synopsis
popupmenu(agent, deflt, relief)
include "popuphelp.g"

Description
`popupmenu` Is a function which adds to an agent a "?" button with popup selection options of "none", "both", "mb3" or "hover". By specifying a deflt argument of 0, 1, 2 or 3 you can select one of them as the default (default both assumed).

Arguments

agent	agent to which menu button should be added Allowed:agent
deflt	default popup useage type Allowed:integer Default: F
relief	button relief Allowed:relief type string Default: flat

Returns
T (or fail)

Example
See "popuphelp".

1.1.21 addpopuphelp - Function

Package display
Module guiutils

add popup help to a GUI

Synopsis
addpopuphelp(agents, maxlevels)
include "popuphelp.g"

Description
`addpopuphelp` Is a function which recurses through a data structure (typically this will be the "private" variable of your closure tool, and for every agent that it finds that has a field named "shorthelp", it will add the code necessary to add "popup" help to that agent.
It is advised to use the "popuphelp" function directly in stead.
Sometimes record/agent structures are set up where a subrecord is in fact a reference to a parent record. Since this might cause infinite recurses, a `maxlevels` parameter is used to prevent such infinite loops. It is set to 4 by default, which should generally suffice, but very deep structures may require this number to be raised.

Arguments

agents	structure that holds the agents Allowed:record or agent
maxlevels	maximum number of levels to recurse Allowed:4

Returns
T (or fail)

Example

```
include 'popuphelp.g';
private := [=];
private.f := frame(height=100);
private.b := button(private.f);
private.b.shorthelp := 'Press the button';
private.l := listbox(private.f);
private.l->insert('AAA'); private.l->insert('BBB');
private.l.shorthelp := function(name, index) {
    # Only need name here.
    return spaste('Your cursor is near ', name);
}
addpopuphelp(private);
```

1.1.22 barchart - Tool

Package display
Module guiutils

Record values and display as a barchart
include "barchart.g"

Description

`barchart` tools are used to record name and value pairs, and to optionally display them in barchart form in a GUI.
If enough colors are available, the GUI display will use green bars if the size is below a specifiable "large" value, and red if the value is above the large value. If there are not enough colors, the bars are always drawn as outlines in black and white.

Example

```
- bc := barchart('My Title', large=30);
- bc.chart('small', 10)
- bc.chart('large', 50)   # Red, because it is >30
- bc.chart('small', 20)   # Update with new value
- bc.large(15)            # Turn small to Red by changing threshold
```

See Also

1. Glish/PGPLOT
2. pgplotter
3. pgplotwidget

barchart - Constructor

Package	**display**
Module	**guiutils**
Tool	**barchart**

Create a barchart tool

Synopsis
barchart(title, width, height, large, show)

Description
Note that after construction of the barchart, even if `show` is `T`, the barchart will not be visible on the screen until at least one value is charted.

Arguments

title	title for the barchart frame Allowed:string Default: AIPS++ Bar Chart
width	width of the barchart in pixels Allowed:integer Default: 400
height	height of the barchart in pixels Allowed:integer Default: 200
large	charted values larger than this are displayed in red if enough colors are available Allowed:real Default: 1.0e+38
show	show charted values? Allowed:boolean Default: T

Returns
barchart tool or fail

barchart.chart - Function

Package	**display**
Module	**guiutils**
Tool	**barchart**

Chart a new name,value pair

Synopsis
chart(name, value)

Description
This function adds a new name,value pair to the barchart. It does not matter if the name already exists or not, and value can increase or decrease.
If `name` and `value` are arrays, you can update many values at once (of course the arrays must be the same length).

Arguments

name	name of the barchart item Allowed:string
value	new value for "name" Allowed:real

Returns
T on success or F on error

---

barchart.remove - Function

Package	**display**
Module	**guiutils**
Tool	**barchart**

Remove an item from the barchart

Synopsis
remove(name)

Description

This function removes a named item from the barchart.

Arguments

name	name of the barchart item Allowed:string	

Returns
T on success or F on error

---

barchart.values - Function

Package	**display**
Module	**guiutils**
Tool	**barchart**

Return all values from the barchart as a record

Synopsis
values()

Description
It is occasionally useful to retrieve all the values from a barchart. This function allows you to do that.

Returns
record of name=value pairs

---

barchart.nogui - Function

Package	**display**
Module	**guiutils**
Tool	**barchart**

Do not display the barchart any more (until gui() is called)

Synopsis

nogui()

Description
This function removes the barchart GUI from the screen if it is being displayed, and prevents it from being displayed until `gui()` is called.

---

barchart.gui - Function

Package	**display**
Module	**guiutils**
Tool	**barchart**

Display the barchart if possible

Synopsis
gui()

Description
This function causes the barchart to be displayed in a GUI if possible (that is, if the glish builtin `have_gui()` is `T`. If enough colors are available (4) a color display is used, otherwise white on black.

---

barchart.large - Function

Package	**display**
Module	**guiutils**
Tool	**barchart**

Display and optionally modify the "large" value

Synopsis
large(newval)

Description

If called with no arguments, this function merely returns the current value of the "large" value (the value at which bars will be drawn in red if enough colors are available).

If called with a value, the large value will be changed as well as returned. If the GUI is being displayed the color of displayed bars will be changed to red if necessary.

Arguments

newval	new large value Allowed:real Default: unset

Returns
real (large value)

---

barchart.done - Function

Package	**display**
Module	**guiutils**
Tool	**barchart**

Delete the barchart tool from memory

Synopsis
done()

Description
This function removes the barchart GUI from memory. After calling `done`, the tool is destroyed and cannot be accessed.

1.1.23 choicewindow - Function

Package	**display**
Module	**guiutils**

Display a question in a window with multiple choice answers

Synopsis
choicewindow(question, choices, interactive, timeout, ws)

include ”choicewindow.g”

Description
The choicewindow is a simple dialog window that lets one pose a question to a user with a limited number of responses. The window will dismiss itself after the timeout period specified returning the first choice in the choices lists. If the user make a selection before the timeout it will return the user's choice.

Arguments

question	The question you want answered Allowed:string
choices	Vector of choices Allowed:string
interactive	Is the session iteractive Allowed:Boolean Default: has_gui()
timeout	specifies the number of ticks in .2s for the window to automatically dismiss itself Allowed:integer Default: 150
ws	widget server Allowed:agent Default: dws

Returns
T (or fail)

Example

```
include 'choicewindow.g';
cw := choicewindow('Do you want to continue?', "Yes No");
```

1.1.24 infowindow - Function
Package display
Module guiutils

Display an informational message in a window