gadgets

COLLABORATORS			
	TITLE : gadgets		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 24, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

gadgets

1.1 GUIEnvironment/Gadgets guide

GUIEnvironment

Gadgets guide

==========

 Carsten Ziegeler
Augustin-Wibbelt-Str.7
D-33106 Paderborn
Germany

==========

1.2 Creating gadgets

Creating gadgets using GUIEnvironment is very simple. First you have to
create the GUIInfo structure !
After that simply do for each gadget a call to CreateGUIGadget with
the needed attributes and then you can use these gadgets after a call
to DrawGUI.

CreateGUIGadget needs first the position and the size of the gadget.
The position is always inside the window ! So a position of 0,0 would
be the left top corner of the inner window !

If the gadget is created GUIEnvironment sets the UserData entry of this
gadget to a GUIGadgetInfo structure. This structure is read only and
it is not allowed to change the UserData entry ! The GUIGadgetInfo
structure also provides the possibility of own user data !

Usually the textFont of the GUIInfo structure is used for the gadgets.
But it is also possible to override this setting by using the GEG_Font tag.

It is not allowed to specify the gadget's ID. GUIEnvironment gives the
first gadget the ID 0, the second the ID 1 and so on !

With the GEG_CreationAHook you can define a hook function for this gadget
which is called every time the gadget is created/redrawn ! This is usefull
for gadtools GENERIC_KIND gadgets. GUIEnvironment first creates the gadget
and then calls your hook function with a pointer to the gadget in the A2
register. If your hook function is not able to do its work, you have to
return FALSE, and the creation is stopped, otherwise return TRUE.

Changing gadgets must be done using GUIEnvironment. This includes such
things as enabling/disabling gadgets !

SEE ALSO
     Gadget description flags
     Gadget creation hook

1.3 Creating Gadtools gadgets

CreateGUIGadget does the following replacements for you for the gadget's
width and height:

```
  CHECKBOX_KIND : if width  = 0 then width  = GadTools.checkboxWidth
                : if height = 0 then height = Gadtools.checkboxHeight
  MX_KIND       : if width  = 0 then width  = GadTools.mxWidth
                  if height = 0 then height = GadTools.mxHeight
```

```
  STRING_KIND   : if height = 0 then height = gadget font->ySize + 4
  INTEGER_KIND  : the same as STRING_KIND

If you use gadget text for a gadtools gadgets you have to use the
GEG_Flags tag to define the gadget's text place. Remember to set this
tag for each gadtools gadget with text !

You can use every gadtools tag to define the gadget.

GUIEnvironment also offers a notify function for gadtools gadgets.
This means you tell GUIEnvironment that a special variable belongs
to this gadget and every time a message for this gadget arrives,
the variable is updated and contains the new value/state of the
gadget.
For this function you have to use the GEG_VarAddress tag. It
replaces for each gadtools gadget a different tag as stated below.
Don't use the replaced tag together with GEG_VarAddress:

STRING_KIND  : GEG_VarAddress replaces GTST_String
INTEGER_KIND : GEG_VarAddress replaces GTIN_Number, but this tag now
               must contain the address of a LONG variable !
MX_KIND      : GEG_VarAddress replaces GTMX_Active, but the tag data
               is a pointer to a UWORD variable !
CYCLE_KIND   : GEG_VarAddress replaces GTCY_Active, but the tag data
               is a pointer to a UWORD variable !
CHECKBOX_KIND: GEG_VarAddress replaces GTCB_Checked, but the tag data
               is a pointer to a boolean UBYTE variable !
SLIDER_KIND  : GEG_VarAddress replaces GTSL_Level, but the tag data is
               a pointer to a WORD variable !
SCROLLER_KIND: GEG_VarAddress replaces GTSC_Top, but the tag data is
               a pointer to a WORD variable !
LISTVIEW_KIND: GEG_VarAddress replaces GTLV_Selected, but the tag data
               is a pointer to a UWORD variable !
PALETTE_KIND : GEG_VarAddress replaces GTPA_Color, but the tag data
               is a pointer to a UWORD variable !

SEE ALSO
     Gadget message handling
```

1.4 BOOPSI gadgets

```
Using BOOPSI gadgets is very simple. You can define them in the same
manner you do it with gadtools gadgets.
You can specify any tag for this gadgets except the following ones
which are set by GUIEnvironment: GA_ID, GA_UserData, GA_Left, GA_Top,
GA_Width, GA_Height, GA_Previous, GA_DrawInfo.
GUIEnvironment also sets the GA_Text tag with the value of the GEG_Text
gadget (or the localized string !).
```

1.5 The GUIEnvironment gadgets

GUIEnvironment offers some own gadget kinds which simplify designing
a GUI.
But this gadgets are not real gadgets, so don't use the gadget
structure created by GUIEnvironment for these gadgets ! Use only the
functions provided to access these gadgets !
The only valid entry of the gadget structure is the UserData entry !

These gadgets never cause any message and it is also possible to draw
some gadgets inside them !

Creating this gadgets follows the same rules as for gadtools gadgets !

1.6 Gadget description flags

If you want to make your GUI resizable you have two possibilities:
a) Each time a IDCMP_NEWSIZE message arrives, you calculate the new sizes
   and positions by hand and then do the resizing with this new values !
b) You use the gadget description flags together with GUIEnvironment.

Using methode a) is very difficult for you. For this GUIEnvironment
offers the possibility of resizing gadgets. See the chapter about
modifying gadgets for more information !

The gadget description flags over an object orientated method of defining
gadgets.
For each gadget you have to use the GEG_Description tag. The data of
this tag says GUIEnvironment how to use the values for the gadget.
This means, you say this gadget has a distance to that gadget of 10. In
the current version you can specify the distance to other gadgets or to
the window borders.
To make the calculation easier you have to define the GUI for a certain
window size ! The GUI for other window sizes is calculated out of the
new window size in proportion to the certain one !
When now the size of the window changes, GUITools can determine, using
these fields, how far the size has changed and recalculate the gadgets
If you want to open your window in a size the user can define, this can
easily done in the following way:
- Open the window with the user defined size
- Create the GUIInfo structure with the GUI_CreationWidth and GUI_Cre-
  ationHeight tags which tell GUIEnvironment the size the GUI was
  designed for !
The resizing for the user defined window size is then done by GUIEnvironment
when you call DrawGUI.

But now the hardest part, the defining of the gadget's positions and
sizes ! (If you don't understand this reading the first time, don't
give up, because my explanations are not as clearly as I wanted them
to be, but take also a look at the demos for resizable gadgets and
perhaps you will know what GUIEnvironment can do !)
For every gadget attribute (left edge, top edge, width and height)
GUIEnvironment uses a description byte. This byte consists of three parts:
1.) The distance kind:
      GEG_DistNorm This is the normal distance, that means the attribute
                    is handled in the same way as without this byte. This

```
                distance is for constant positions and sizes.
      GEG_DistAbs   This specifies the distance to a given object. The size
                    of the window is not regarded. The distance will always
                    be the same.
      GEG_DistRel   This distance is also a distance from a given object.
                    But if the window size changes, the distance will change
                    in the same way.
      GEGD_DistPercent In the first way this does the same as GEG_DistNorm,
                    but if now the window size changes, the distance will
                    change in the same way.

2.) The object kind:
      If you use GEG_DistAbs or GEG_DistRel GUIEnvironment needs also the
      kind of the object the distance is to:

      GEG_ObjBorder    this is the window border
      GEG_ObjGadget    the distance to another gadget

3.) The object part

      If you specify the distance to another object, you also have to say
      to which part of the object.
      This can be GEG_ObjRight, GEG_ObjLeft, GEG_ObjBottom or GEG_ObjTop

And that's all ! Really simple, isn't it. But how can you tell GUIEnv
these descriptions. For this reason there is the GEG_Description tag.
Because every tag has a data value of 32 Bits, it is possible to store
the information for all four attributes into one tag !
The highest byte contains the left edge, the next byte the top edge,
the next byte the width and the lowest byte the height of the gadget !

To build this information data, you can use the GADDESC macro !
If you specify an attribute's distance to another gadget, normally the
previous created gadget is used. If you want to define the distance to
a different one, you have to use the GEG_Objects tag.
It has the same structure as the GEG_Description tag except that
every byte represents a gadget number. But you can only use gadgets
which are already created !

For more information refer to the included example source files !
Here follows a short example: You want to create a gadget, which is
10 points from the left and the top border and which right border is 20
away from the left border of the first created gadget (gadget number 0)
and the top border has a distance of 10 points to the second created
gadget (with gadget number 2):

    CreateGUIGadget(GUI, 10, 10, -20, -10, ANYKIND,
                    GEG_Description, GADDESC(GEG_DistAbs + GEG_ObjBorder +  ↩
                       GEG_ObjLeft,
                                             GEG_DistAbs + GEG_ObjBorder +  ↩
                                                GEG_ObjTop,
                                             GEG_DistAbs + GEG_ObjGadget +  ↩
                                                GEG_ObjLeft,
                                             GEG_DistAbs + GEG_ObjGadget +  ↩
                                                GEG_ObjTop),
                    GEG_Objects, GADOBJS( 0, 0, 0, 1), NULL);
```

GADOBJS is a macro to build the data for the GEG_Objects tag.

1.7 Localizing gadgets

Localizing the gadgets is very easy:

First, when creating the GUIInfo structure, you have to specify the
name of the catalog file for this GUI and the number of the first
gadget text within this catalog.
The catalog should be designed in that way, that the gadgets' texts
are in straight order.
For example: A catalog for the three button gadgets could look like
this:

100: Load
101: Save
102: QUIT

Using the GUI_GadgetCatalogOffset tag with the data of 100 and the
GUI_CatalogFile tag with the file name of the catalog, you don't
need to do anything else. The gadgets will appear in the correct
language (if a catalog exists).
If the Save and QUIT texts would have the numbers 102 and 101
you would have to use the GEG_CatalogString tag for both gadgets.
For Save you would pass GEG_CatalogString, 102 and
for QUIT GEG_CatalogString, 101.
With each call to CreateGUIGadget the catalog string number
is increased by one. You could check the gadgetCatalogOffset
entry of the GUIInfo structure for the current value.

1.8 The gadget help function

If the AmigaGuide is installed and if you have turned on intuitions
gadget help function, GUIEnvironment displayes the node specified in
the GUIGadgetInfo structure for each gadget if a IDCMP_GADGETHELP
message arrives.
Usually CreateGUIGadget creates the node names for you, this means
every gadget gets a node labelled GADGET followed by the gadget ID,
this means the first gadget gets the node name GADGET0, the second
GADGET1 and so on.
If you want to have different node names, use the GEG_GuideNode tag.
The application will get a IDCMP_GADGETHELP message, if the help
function was activated outside the window (msgGadget == NIL) or
inside the window, but not over any gadget (msgGadget == &Window).
Also, the help messages for system gadgets are passed to application.
The msgGadNbr field will then contain the gadget type !

SEE ALSO
     The GUIGadgetInfo structure
     The gadget tags

1.9 Special gadget features

```
For gadtools gadgets GUIEnvironment offers the possibility of notification.
This means, you give a gadtools gadget a variable and GUIEnvironment
updates this variable with the state of the gadget with every message !


Chaining gadgets

GUIEnvironment offers the possibility of chaining gadgets. This means
if you have finished giving input to one gadget, the next gadget is
automatically activated. This is very usefull for several text entry
gadgets which can now be chained, so that the user switches between
the gadgets simply by pressing RETURN, so he won't need the mouse to
do so !
You can specify the following flags as data (using GEG_ChainActivation)

  GEG_ChainUpNext : If the gadget gets an up message, activate the next
                    gadget with the up function.
  GEG_ChainUpPrev : If the gadget gets an up message, activate the previous
                    gadget with the up function.
  GEG_ChainDownNext : If the gadget gets an down message, activate the next
                      gadget with the down function.
  GEG_ChainDownPrev : If the gadget gets an down message, activate the next
                      gadget with the down function.

You start the chaining with the GEG_StartChain tag (use 0 as data !),
and you end chaining with the GEG_EndChain tag. If you here specify TRUE
as tag data, the gadgets are chaining in a cycle, this means this last
gadget activates the first gadget and vice versa.


Activation on startup

If you specify the GEG_Activate tag, the gadget is activated when the
GUI is drawn. You can specify GEG_ACTIVATIONUP or GEG_ACTIVATION_DOWN
as tag data. This determines if the up or down function is called.
The activation is usefull for gadtools text entry gadgets which can
be activated on startup, so the user can immediately enter the text
without using the mouse !


SEE ALSO
     Gadtools gadgets
```

1.10 Gadget action

```
The GUIGadgetAction function offers a efficient possibility to do some
work on several gadgets without only one command !
The action tags get the gadget number/ID sa tag data or you can define
GEG_ALLGADGETS as data, if you want to affect all gadgets.
If you e.g. want to disable all gadgets but not the second gadget (ID=1),
you could do this:
```

```
GUIGadgetAction(GUI, GEG_Disable, GEG_ALLGADGETS, GEG_Enable, 1, TAG_END);

The GEG_SetVar and GEG_GetVar tag offer together with the notification
method a very simple way of setting and getting gadgets' states !

SEE ALSO
     The gadget tags
```

1.11 Gadget message handling

```
With the IDCMP_GADGETUP, IDCMP_GADGETDOWN or the IDCMP_MOUSEMOVE message
GUIEnvironment sets the msgClass, msgGadget and the msgGadNbr field of
the GUIInfo structure with the appropriate information !

If you set the notification for a gadtools gadget, the following steps
are done:

    IDCMP message                      action
-------------------------------------------------------------------------

  IDCMP_GADGETUP     CHECKBOX_KIND  Updates the belonging variable. The
                                    msgBoolCode entry will get the state
                                    of the gadget. (It will get the state
                                    even if the notification is turned off)

                     CYCLE_KIND,    Update the belonging variable. Usually
                     SLIDER_KIND,   there will be no need to check these
                     SCROLLER_KIND, messages. It is sufficient to check
                     LISTVIEW_KIND, the values of the variables when they
                     PALETTE_KIND   are required.

  IDCMP_GADGETDOWN   MX_KIND,       Update the belonging variable. Usually
                     SLIDER_KIND,   there will be no need to check these
                     SCROLLER_KIND  messages. It is sufficient to check
                                    the values of the variables when they
                                    are required.

  IDCMP_MOUSEMOVE    SCROLLER_KIND, Update the belonging variable. Usually
                     SLIDER_KIND    there will be no need to check these
                                    messages. It is sufficient to check
                                    the values of the variables when they
                                    are required.

If the gadget wasn't a gadtools gadget with notification, now the
up or down function is called, depending on the incomming message.
If this function returnes TRUE, the application will get the message,
otherwise not !
If the gadgets are chained, now the next/previous gadget will be activated.

SEE ALSO
     Gadget help function
     Chaining gadgets
     Key equivalents
     Gadget event hook
```

1.12 Gadget Key Equivalents

GUIEnvironment supports key equivalents for gadtools gadgets as noted
in the Libaries RKRM ! The only difference is with BUTTON_KIND gadgets:
Usually there's no difference if the shifted key was pressed or the
unshifted, GUIEnvironment distinguishes these facts !

All key equivalents which are letters ranging from 'A' to 'Z' are
handled automatically ! For all other characters you can specify
a hook function: (using the GUI_VanKeyAHook tag)

Example: Imagine, you want to program your own hypertext tool. Despite of
a lot of other gadgets and menus you have a 'Browse _>' and a
'Browse _<' gadget. The first one has the ID 5 and the second
one the ID 6.

Your hook function could look like this:

```
LONG KeyFctDemo(register __a0 struct Hook *hook,
                register __a2 LONG key,
                register __a1 APTR unused)
/* We don't need __saveds , because no global data is used and
   GUIEnvironment sets the A4 register for us ! */
{
   LONG ret;
   ret = GEH_KeyUnknown;
   if (char(key) == '<')
     ret = 5; /* gadget number 5 */
   if (char(key) == '>')
     ret = 6; /* gadget number 6 */

   return (ret);
}
```

If now a IDCMP_VANILLAKEY message appears and the character is a
key equivalent, GUIEnvironment converts this message to a gadget
message ! (This means, the msgClass and msgCode entries are changed,
but not the intuiMsg field !)

The msgGadNbr and msgGadget fields are set also.

For the gadtools gadgets it does the following substitutions:

gadget kind	action
BUTTON_KIND	msgClass = IDCMP_GADGETUP msgCode = 0 If the key was pressed with shift: msgClass = IDCMP_GADGETDOWN
STRING_KIND, INTEGER_KIND	activates the gadget for input msgClass = gagdetDown msgCode = 0
CHECKBOX_KIND	changes the state

```
                    msgClass = IDCMP_GADGETUP
                    msgBoolCode contains the new state

  MX_KIND           choose the next value without shift and with shift
                    the previous one
                    msgClass = IDCMP_GADGETDOWN
                    msgCode  = new entry number

  CYCLE_KIND        choose the next value without shift and with shift
                    the previous one
                    msgClass = IDCMP_GADGETUP
                    msgCode  = new entry number

  SLIDER_KIND,      without shift one position forward, with shift
  SCROLLER_KIND     one backwards
                    msgClass = IDCMP_GADGETUP
                    msgCode  = new position (new level resp new top)

  LISTVIEW_KIND     without shift next entry, with shift the previous
                    one. If no entry wasn't selected yet, without
                    shift the first and with shift the last one is
                    selected.
                    msgClass = IDCMP_GADGETUP
                    msgCode  = new entry number

  PALETTE_KIND      without shift the next colour will be selected,
                    with shift the previous one.
                    msgClass = IDCMP_GADGETUP
                    msgCode  = new colour number

After this substitution the event is handled as a real gadget event,
so see the chapter about gadget message handling for more information
about the things happening now !

SEE ALSO
     Gadget message handling
     The vanilla key hook
```

1.13 Setting and getting gadget attributes

```
GUIEnvironment offers the SetGUIGadget function to set gadget attributes.
You can pass all attributes which are defined in GUIEnvironment for
setting gadgets (See the gadget tag list), all tags for gadtools gadgets
and also the special tags for BOOPSI gadgets.

You can use the GEG_Status tag to disable/enable all gadget kinds !

But don't use the GA_Left, GA_Top, GA_Width, GA_Height tag. Use the
ones GUIEnv offers instead.
If you want to affect more than one gadget use the GUIGadgetAction
function !

With the GetGUIGadget function you can get all above mentioned tags !
But remember you can only get gadtools attributes if you have version
39 or above of the gadtools.library installed !
```

```
With the GEG_Address tag you get a pointer to the gadget structure !

SEE ALSO
     The gadget tags
     Gadget Actions
```

1.14 The GUIGadgetInfo structure

```
The UserData entry of each gadget is set to a GUIGadgetInfo structure.
This structure is read only. The entries can be changed using the
SetGUIGadget function together with the gadget tags.


struct GUIGadgetInfo
{
  APTR userData;

       Use this for own user data.

  LONG kind;

       The gadget kind.

  APTR gadgetClass;

       Pointer to the gadget class if the gadget is a BOOPSI gadget.
       If the class is public this is a string pointer, otherwise this
       is a pointer to a Class structure.

  struct Hook functionUp, functionDown;

       Hook functions called with IDCMP_GADGETUP/IDCMP_GADGETDOWN
       messages.

  STRPTR guideNode;

       Pointer to an AmigaGuide node which is displayed with a
       IDCMP_GADGETHELP message.

};


SEE ALSO
     The gadget tags
```

1.15 The gadget kinds

```
Apart from the gadtools gadget kinds GUIEnvironment does also know the
following gadget kinds:

 GEG_ProgressIndicatorKind
```

This gadget offers a beam, e.g. to show howfar the progress has come.

With this gadget you can use the following tags:

GEG_PIMaxValue

The maximum value of a progress indicator kind. The value of 100 is very useful, because then it is easier to specify the progress in per cent.

GEG_PICurrentValue

The current value of a progress indicator kind. This is the percentage of the progress. It shows how much is done yet.

These tags can be used for creation and modification.
This gadget also provides a gadget text, but it is not allowed to write the text inside the gadget. Because this gadget never sends messages, there is no need of having a key equivalent and so there should not be an underscore in the gadget text !

GEG_BevelboxKind

This gadget shows a bevelled box, which can either be raised or recessed.

When creating you can specify the following tag:

GEG_BBRecessed

Should the bevelled box be recessed or not.

Don't use gadget text together with bevelled box gadgets. Because this is not a real gadget as mentioned above, it is possible to show gadgets inside this gadget ! So you can draw a border around other gadgets !
Using this gadget instead of a real bevelled box, simplifies such things as resizing GUIs and refreshing !
This gadget never causes a message !

GEG_BorderKind

This gadget draws a border.
This gadget supports gadget text which can only be placed above or below !
Because this is not a real gadget as mentioned above, it is possible to show gadgets inside this gadget ! So you can draw a border around other gadgets !
This gadget never causes a message !

GEG_BOOPSIPublicKind

A BOOPSI gadget which uses a public class.

```
 GEG_BOOPSIPrivateKind

    A BOOPSI gadget which uses a private class.
```

1.16 The gadget tags

```
C : Creation
S : Set / Changing
A : Action
G : Gettable

 GEG_Text          C

  The gadget text.

 GEG_Flags         C

  Gadget flags only for gadtools gadgets.

 GEG_Font          C

  The gadget font.

 GEG_UserData      C S

  Own user data.

 GEG_Description   C

  Gadget description tag for resizing.

 GEG_Objects       C

  Gadget description tag for resizing.

 GEG_GuideNode     C S

  The AmigaGuide node to display as help text.

 GEG_UpAHook    C S

  IDCMP_GADGETUP function.

 GEG_DownAHook  C S

  IDCMP_GADGETDOWN function.

 GEG_CatalogString C

  Number of the string in the catalog for the gadget's text.

 GEG_Class         C

  Gadget class for BOOPSI gadgets.
```

```
GEG_VarAddress    C

 Address of a variable for notification of gadtools gadgets.

GEG_HandleInternal  C

 Handle all messages for this gadget internal if possible.

GEG_StartChain    C

 Start chaining gadgets.

GEG_EndChain      C

 End chaining gadgets.

GEG_Activate      C

 Activate gadget every time it is drawn.

GEG_ChainActivation  C

 Chain activation flags.

GEG_CreationAHook  C

 Gadget creation hook.

GEG_PIMaxValue    C S G

 Progress indicator kind: Maximum value.

GEG_PICurrentValue  C S G

 Progress indicator kind: Current value.

GEG_BBRecessed      C G

 Bevelled box kind: Recessed or not.

GEG_Disable         A

 Action tag: Disable gadgets.

GEG_Enable          A

 Action tag: Enable gadgets.

GEG_SetVar          A

 Action tag: Set gadgets to variable contents (needs notification)

GEG_GetVar          A

 Action tag: Get variable values from gadgets (needs notification)

GEG_ActivateUp      A
```

```
  Action tag: Call up function

 GEG_ActivateDown    A

  Action tag: Call down function

 GEG_Address         G


 GEG_LeftEdge        S G

  Gadget left edge.

 GEG_TopEdge         S G

  Gadget top edge.

 GEG_Width           S G

  Gadget width.

 GEG_Height          S G

  Gadget height.

 GEG_Redraw          S

  Boolean tag: Redraw the gadget immediately if the new size/position
               is set. (Default: TRUE)
  If you set this to FALSE, you can first define new sizes for all gadgets
  and then redraw/resize them using DrawGUI.

 GEG_Status        C S G

  This is a replacement for the intuition GA_Disabled tag. But this tag
  works for ALL gadgets and the data specifies if the gadget is active
  and not if it is disables as the GA_Disabled tag does !
```

1.17 rcs

```
$RCSfile: Gadgets.guide $

$Revision: 1.5 $
    $Date: 1994/11/03 15:50:38 $

  GUIEnvironment Gadget Guide

  Copyright © 1994, Carsten Ziegeler
                    Augustin-Wibbelt-Str.7, 33106 Paderborn, Germany
```