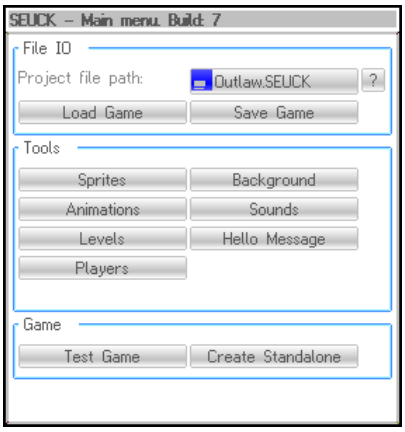
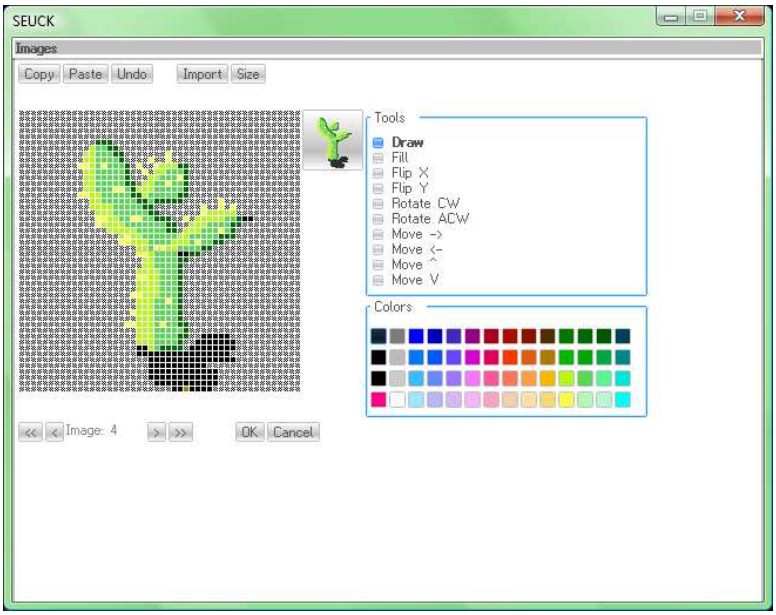
DDgui – User manual (2009-12-15)

Dialogs

DDGui ist the GUI toolkit for the programming language GLBasic. More Information can be found on www.glbasic.com.
It offers a slim set of controls, called widgets, that are required for simple GUI programs. Here's a few screenshots of DDgui dialogs.



[A simple main menu dialog with buttons]



[a quite complex image editor]

DDgui is based on "dialogs", that is main widgets, where all other widgets are placed upon. In DDgui, only one dialog can be used. Dialogs lower on the stack can be displayed, though. They are greyed out then.

Make a project

First thing you do it to create a project, then add the DDgui.gbas file from the Samples/common directory to your project. Just use the "file" tab at the right side of the GLBasic editor to do this.

DDgui_pushdialog – Start a dialog

With ddgui_pushdialog you create a new dialog widget on the DDgui widget stack. There's nothing shown, yet. You just give the position and size of the dialog at initialization time:

```
DDgui_pushdialog(0,0,320,240)
```

DDgui_show – show what you have got

The command DDgui_show not only displays the current (and optionally all older) dialogs, it also handles all the widgets on the dialog. After a call to ddgui_show, you can check if buttons have been pressed or controls have changed state.
Usually a main loop for a ddgui dialog looks like this:

```
WHILE TRUE
      DDgui_show(FALSE);
      SHOWSCREEN
      // check for user interaction
WEND
```

Widgets – the flavour of GUI

There's a small variety of widgets to use in DDgui. You can extend DDgui to use your own widgets, however, too. Each widget has a unique ID. If you do not provide an ID (pass an empty string), DDgui will name it automatically. You must provide an ID if you want to interact with the widget, however.
The widgets are aligned side by side from left to right. If there's not enough space for a widget, it will wrap to the next line. The height of a line is determined by the most high widget in the line. DDgui provides spacer elements to align widgets nicely, as well as frames for grouping elements and changing the line break behaviour.
Each widget has a width and a height parameter. If you leave these 0, the widget will choose a proper size to display the contents you specified at initialization time. You can resize images by changing the "WIDTH" and "HEIGHT" parameters. Which lead us to:

Properties

Each DDgui widget has special properties. You can set these with:

```
DDgui_set("id_of_widget", "PROPERTY_NAME", value$)
```

and get the value of a property with:

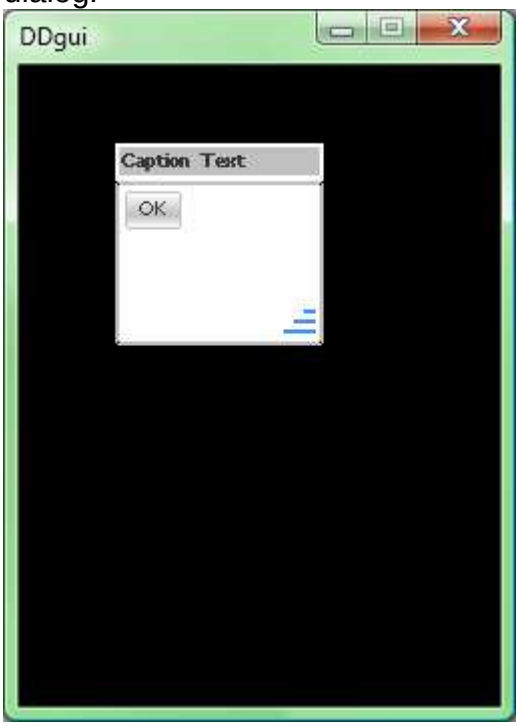
```
DDgui_get$("id_of_widget", "PROPERTY_NAME", value$)
```

Pay attention, there's also a DDgui_get function (no $) which returns the property directly as a number, so you can avoid convertions.
The available property names vary from widget to widget. Read the description for each widget for more details.

The widget "" is a dummy widget that accesses the main dialog itself. So, for the dialog's width you query DDgui_get("", "WIDTH").
The main dialog's caption can be optionally set with the TEXT property:


```
DDgui_pushdialog(10,10,200,200)
DDgui_set("", "TEXT", "Caption Text")
DDgui_button("bt_ok", "OK", 0,0)
```

If you want a dialog moveable, set the MOVEABLE property to TRUE. With SCALEABLE you can have a gripper in the bottom right corner that allows resizing a dialog.



[scaleable dialog]

Static text

You can make static text boxes with DDgui_widget. Just pass an ID, that is a unique string, to the widget, set some initial text and provide a width and height. If the text is

too wide, it will be word wrapped into the box. A widget will also become taller if the text gets longer. So the text will always be visible.

```
DDgui_UpdateFont(TRUE) // use font kerning
DDgui_pushdialog(10,10,200,200)
DDgui_widget("id_text", "Here's some text for you", 0,0)
WHILE TRUE
      DDgui_show(FALSE)
      SHOWSCREEN
WEND
```

A widget has the properties:
WIDTH, HEIGHT, TEXT, CLICKED
The first 2 are quite obvious. TEXT is what is displayed. If you click and release the mouse button on the widget, the CLICKED property will become "1" once after a DDgui_show().

Buttons

A button is similar to a widget, it's only not word wrapping the text, and it's changing its colour when the mouse hovers over it.

```
DDgui_pushdialog(10,10,200,200)
DDgui_button("id_text", "Here's some text for you", 0,0)
WHILE TRUE
      DDgui_show(FALSE)
      SHOWSCREEN
WEND
```

A button can also have an image attached. For this use "SPR_Bxxx" instead of the button text. The number "xxx" is the ID of a sprite you loaded with LOADSPRITE.
A button also can have a colour, so you can choose colours with buttons. For this use "SPR_Cxxx" for the TEXT property. In this case "xxx" is a number created with the RGB function. If you click an image button, a colour chooser dialog pops up. If you don't want this, set the READONLY property to TRUE for this button.
READONLY buttons will have a different look and will not generate a CLICKED property change when clicked.

Sliders

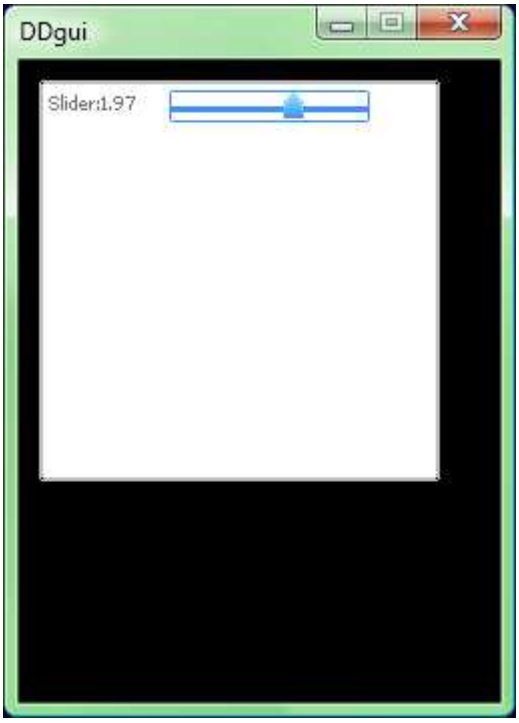
A slider is a left-right control, that can have any value range.

```
DDgui_pushdialog(10,10,200,200)
DDgui_widget("id_text", "Slider:999", 0,0)
DDgui_slider("id_slider", 0,0,0)

DDgui_set("id_slider", "MINVAL", 1.0)
DDgui_set("id_slider", "MAXVAL", 2.5)

WHILE TRUE
      DDgui_show(FALSE)
      slider_value = DDgui_get("id_slider", "TEXT")
      DDgui_set("id_text", "TEXT", "Slider:"+slider_value)

      SHOWSCREEN
WEND
```

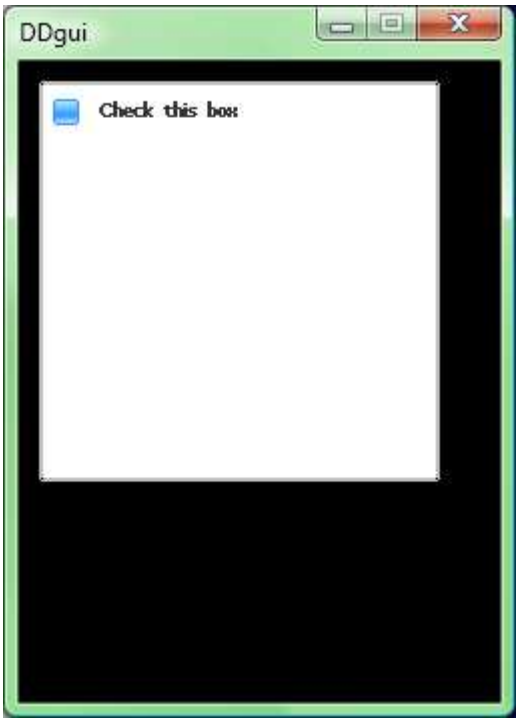


You see, we're getting quite sophisticated here. The slider has these properties: WIDTH, HEIGHT, TEXT, MINVAL, MAXVAL.
MINVAL and MAXVAL are the bounds of the left and right value of the slider. By default it's set to 0 and 1.
The TEXT is the numerical value of the slider position.

Toolbar

The toolbar command is just a macro to quickly create a set of buttons. You just give an array of IDs and a 2nd array that has indices for sprite image IDs to show for each of the buttons.

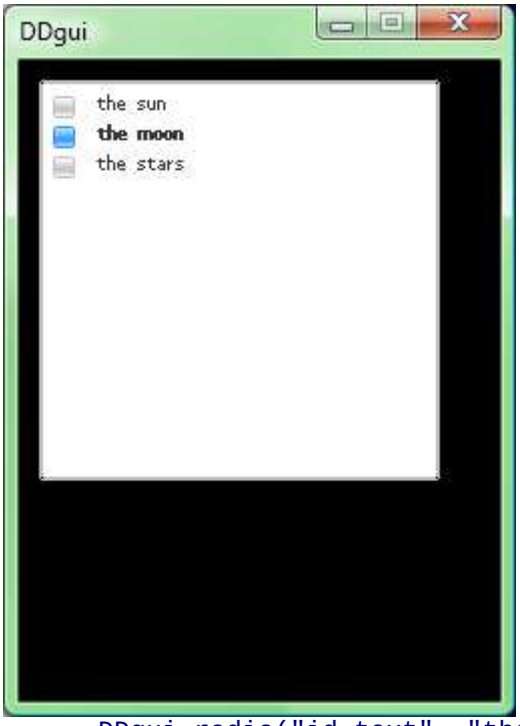
Checkbox




```
DDgui_pushdialog(10,10,200,200)
DDgui_checkbox("id_text", "Check this box", 0,0)
```

A checkbox is a button, that has a selected state. You will get a CLICKED property change when you click it. It has the SELECT property set to TRUE or FALSE depending on the checkbox state.

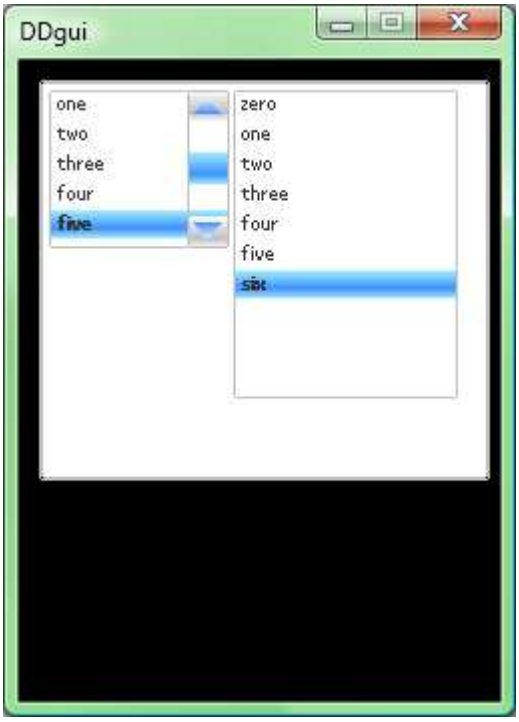
Radio buttons




```
DDgui_radio("id_text", "the sun|the moon|the stars", 0)
```

A radio button is a widget that has a set of checkboxes, but only one of them can be active at a time. The buttons are aligned one below the others and treated as a solid block for line wrapping of the widgets. The SELECT property will indicate which button is set. Where 0 is the first item and so on. If no item is selected, the SELECT property will be negative.
The COUNT property gives information about the number of radio buttons in this group. You will also get the CLICKED notification property change.
The TEXT property is a set of the radio buttons' texts, separated by a binary or operator "|". Example: "the sun|the moon|the stars".
The creation function for radio buttons has no height parameter, since all buttons are always shown.

Lists

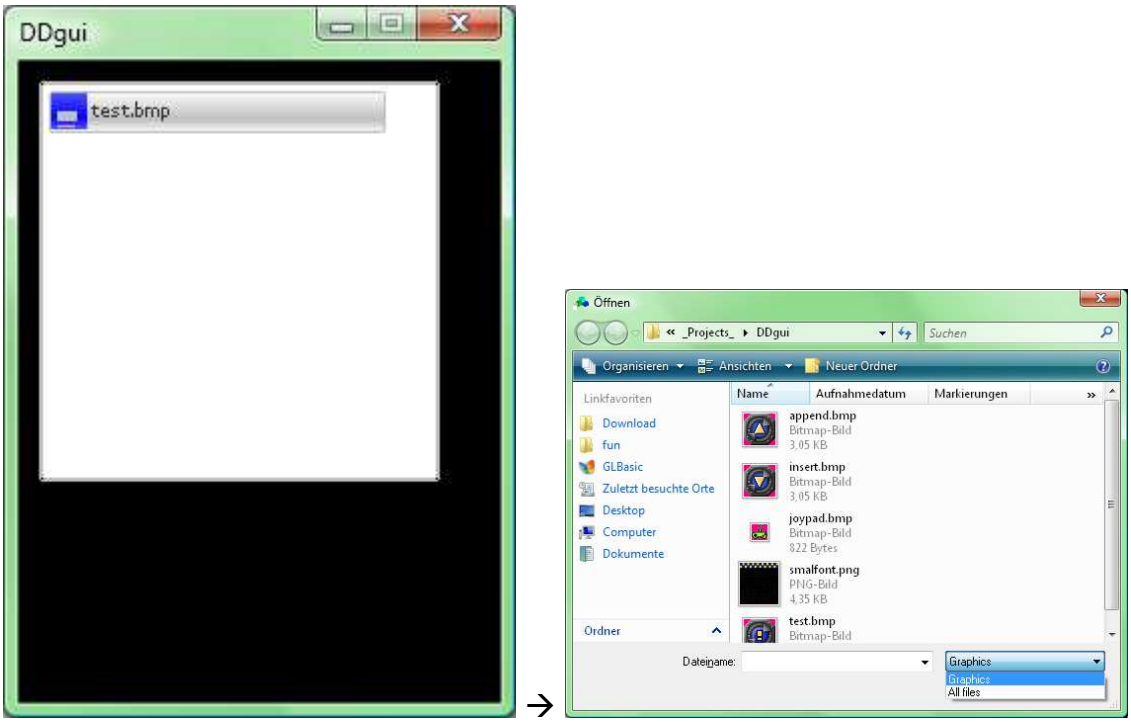


[two list boxes. One with scrollbar, one tall enough to fit the contents]

```
DDgui_pushdialog(10,10,225,200)
DDgui_list("id_list1", "zero|one|two|three|four|five|six", 90,80)
DDgui_list("id_list2", "zero|one|two|three|four|five|six", 0,160)
```

A list is completely the same as a radio button group, but it allows to specify a height. If not all items can be displayed in the list, a scrollbar will be shown. You must, however, provide a height when creating the list box.

File




```
DDgui_pushdialog(10,10,200,200)
DDgui_list("id_text", "test.bmp", _
           "Graphics|*.bmp;*.png|All files|*.*", 0,0)
```

The file widget is a button with a disk symbol attached. If you click it, a file browser will open that lets you click on a file. You will get a CLICKED property change for that. The TEXT property yields the path name to a file. If can be a relative path name or an absolute one.
Additionally to the button's properties, the file offers a FILTER property, which lets you set and get the filter string as you pass it to FILEREQUEST$() in GLBasic. Leaving the filter empty will result in a "*.*" filter.

Texts

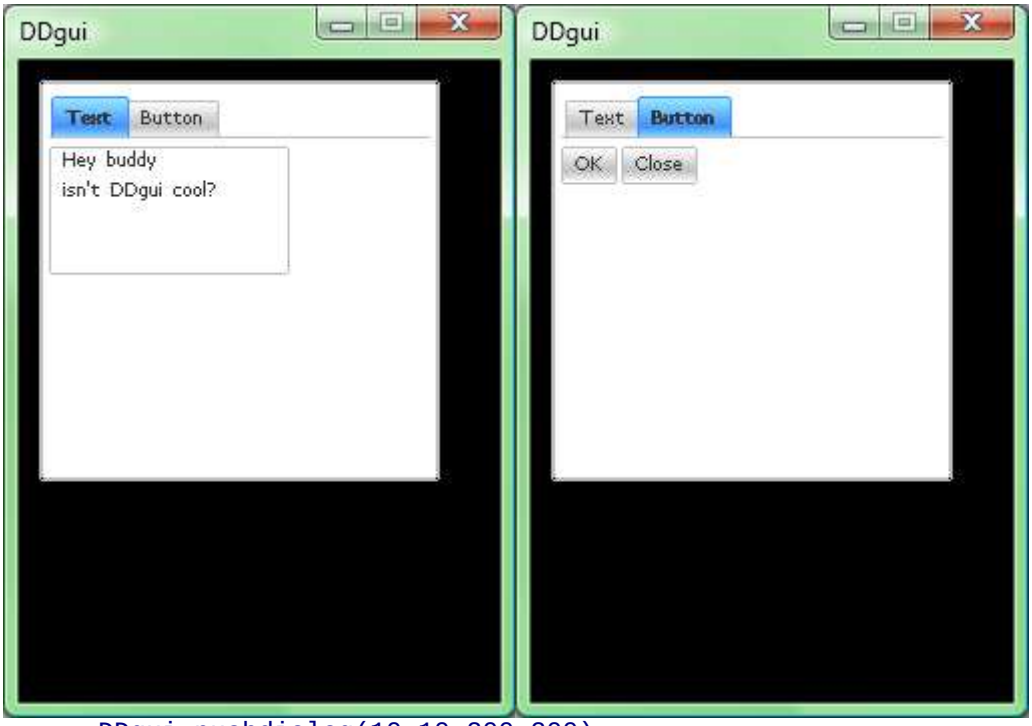

```
DDgui_pushdialog(10,10,200,200)
DDgui_text("id_text", "Hey buddy\nisn't DDgui cool?", 120,64)
DDGUI_AUTO_INPUT_DLG = TRUE
```

The text widget is probably the most powerful widget in DDgui. It offers a text box with word wrapping, that can have selection, a caret to edit the text and a vertical scrollbar if required.
You have the properties SELSTART and SELEND, which are indices in the TEXT property for the caret position and current selection.
It supports the READONLY property, too.
On GP2X and PocketPC you will get into an edit dialog when you tap on such a widget. You can force and suppress this behaviour by setting the global variable DDGUI_AUTO_INPUT_DLG to TRUE or FALSE after a call to DDgui_pushdialog().

[DDgui keyboard for texts on PocketPC/GP2X]
You can have this keyboard with the command:

```
text$ = DDgui_input$("test test")
```

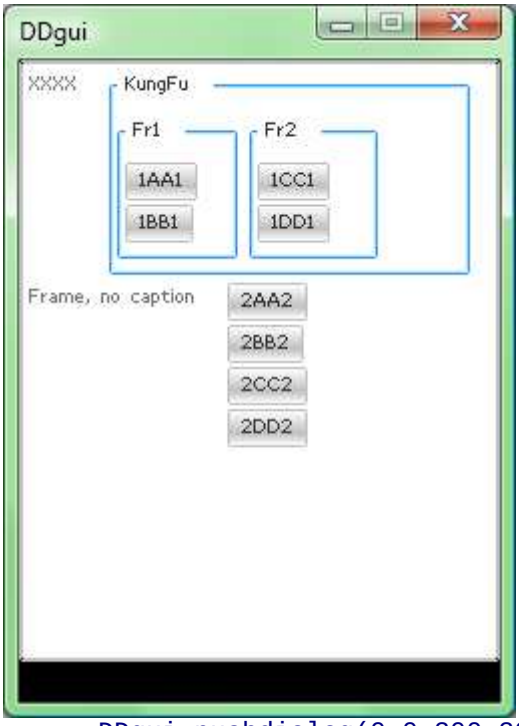
Tab books




```
DDgui_pushdialog(10,10,200,200)
DDgui_tab("id_tab", "Text,id_text|Button,bt_ok,bt_close")
DDgui_text("id_text", "Hey buddy\nisn't DDgui cool?", 120,64)
DDgui_button("bt_ok", "OK", 0,0)
DDgui_button("bt_close", "Close", 0,0)
```

A tab book is a horizontal aligned set of buttons that allow you to enable or disable the visibility of certain widgets. If you want to disable widgets in a frame, it's not enough to specify the frame alone, yet.
Also make sure that spacers are no problem, since they cannot be hidden, yet.
The tab book always covers the whole width of a dialog. It's possible to have more tabs, though.
The TEXT property of a tab is a bit more complex compared to other widgets. First, there's the name to display for a tab page. Then, separated by commas, a list of widget IDs to show for this tab page. The widgets may not have been created, yet. Then each tab is separated from the next one by a binary or operator "|". You can use a frame as a widget. All childs inside the frame are hidden or show then, when you click on the tab page.
You get a CLICKED notification property change when the user clicks on a tab, and you can get the currently active tab with the SELECT property.
If you want to select a TAB, please use the DDgui_selecttab(id$, isle%) function.

Frames




```
DDgui_pushdialog(0,0,300,300)

DDgui_widget("st1", "XXXX",0,0)
DDgui_framestart("fr1", "KungFu", 180)
      DDgui_framestart("fr1_1", "Fr1", 60)
            DDgui_button("1aa1", "1AA1",0,0)
            DDgui_button("1bb1", "1BB1",0,0)
      DDgui_frameend()
      DDgui_framestart("fr2", "Fr2", 64)
            DDgui_button("1cc1", "1CC1",0,0)
            DDgui_button("1dd1", "1DD1",0,0)
      DDgui_frameend()
DDgui_frameend()

DDgui_spacer(10000,0)
DDgui_widget("1zz1", "Frame, no caption",0,0)

DDgui_framestart()
```

```
        DDgui_button("2aa2", "2AA2",0,0)
        DDgui_button("2bb2", "2BB2",0,0)
        DDgui_button("2cc2", "2CC2",0,0)
        DDgui_button("2dd2", "2DD2",0,0)
    DDgui_frameend()
```

A frame is a widget that can contain other widgets. They line-wrap within the frame. If a frame has no TEXT property, it will be drawn invisible. Otherwise it has a border and a caption text.
If a frame has no width at the beginning, it will use the widest child widget's width plus an optional border width at the first DDgui_show() call. You can manually change the width later, too. The height adjusts so that it's the smallest possible rectangle around the frames widgets.
You create a frame with DDgui_framestart. Then all subsequently created widgets are placed into this frame until you call the corresponding DDgui_frameend() function. You can have nested frames, too.
You can refer to a frame in a DDgui_tab, which makes programming tab pages even quicker.

Helper functions

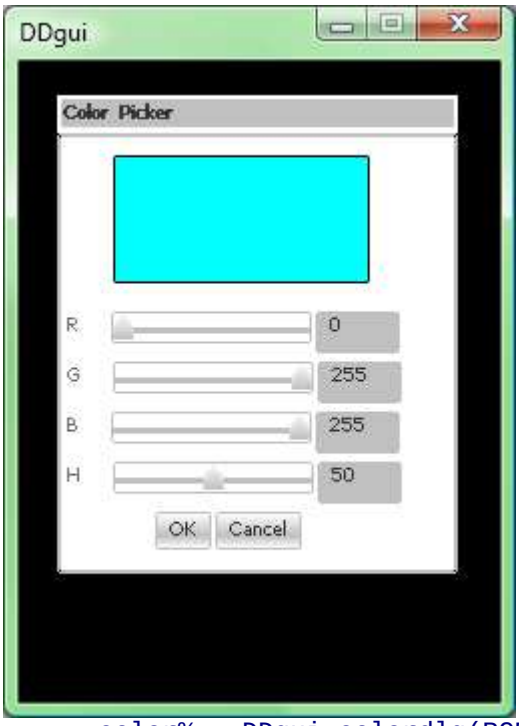
DDgui offers a few quite interesting helper functions to make working with DDgui even easier.

DDgui_msg


```
    DDgui_msg("Hello World", TRUE)
```

Just opens a message box with an optional Yes/No button set, or just an OK button.

DDgui_colordlg




```
color% = DDgui_colordlg(RGB(0,255,255))
```

This function opens a color dialog, as you get one when you have a button with a colour instead of a text (SPR_Cxxx). The return value of this function is the chose colour.

DDgui_getitemtext$

With this function you can get the nth item text of a list box or a radio group.

DDgui_insertitem

You can easily insert items in between a list box or a radio group with this function.

DDgui_deleteitem

When you want to delete an item from a list box, please use this function.

DDgui_input$

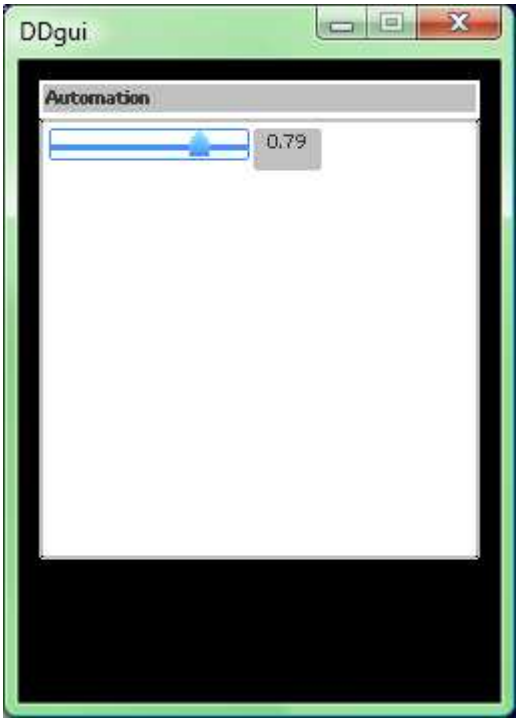
As indicated at the text widget, this opens a on screen keyboard that can be used with a touchscreen device. The resolution should be 240x240 or above.

DDgui_hide

Call this function to hide or show a widget. A hidden widget will not be updated nor drawn.

DDgui_automate

If you want to transport properties from one widget to another one, this function is your friend. You can have DDgui automatically DDgui_get$ some property and DDgui_set it to some other property. A very good example is a slider and a text box, that always displays the slider's value.



[DDgui automation example screenshot]

```
DDgui_UpdateFont(TRUE) // use font kerning

DDgui_pushdialog(10,10,220,220)
DDgui_set("", "TEXT", "Automation")
DDgui_slider("sl_1", 0.5, 0,0)
DDgui_text("tx_val", "xxxx", 0,0)
DDgui_set("tx_val", "READONLY", TRUE)
DDgui_automate("sl_1", "TEXT", "tx_val","TEXT")

WHILE TRUE
      DDgui_show(TRUE)
      SHOWSCREEN
WEND
```

Error handling and debugging

For debugging purposes, turn the debugger icon in the editor on. Then run your DDgui dialog. You will get any of these error messages:

- DDgui_get: No active dialog!
  There's no dialog present. You cannot DDgui_get/set without a dialog.
- DDgui_get: Widget not found xxxx
  You try to get properties from a widget with a non-existing ID. This is usually a typo problem.
- DDgui_get$: Widget property TXET is unknown
  You have a typing mistake in the property of a widget. Make sure you only use the properties mentioned above for DDgui. Even for user defined widgets.

Appearance

Colours

You can change the colours of a DDgui dialog by changing the properties COL_BRIGHT, COL_NORM, COL_HOVER_BRIGHT and COL_HOVER_NORM to the desired RGB values.

COL_BRIGHT	bright colour
COL_NORM	normal colour
COL_HOVER_BRIGHT	bright colour when under mouse
COL_HOVER_NORM	normal colour when under mouse

There's a function: Colorize() in the DDgui sample project, that allows you to change the colours in a DDgui dialog and then write the source code for these colours to a file.

Fonts

DDgui by default uses the current font, set with SETFONT. If you want to have variable width fonts, please call DDgui_UpdateFont(TRUE) once. This creates an internal kerning table to display fonts nicely.
Take a look at that function to see how you can do this very fast without GETPIXEL calls. On the iPhone, however, this function does take about a few seconds for a large font.

Callbacks

FUNCTION DDgui_backrect: x%,y%,dx%,dy%, col%

This function can be overridden to draw a small frame around buttons and other widgets. By default it draws a 1 pixel wide rectangle with rounded borders.

FUNCTION DDgui_backgnd: col1%, col2%, x%, y%, dx%, dy%

With this function you can override the drawing behaviour of DDgui filled rectangles. DDgui by default provides 2 ways depending on whether you have OpenGL hardware support or not.

User defined widgets

You can have user defined widgets, easily. Just call DDgui_userdefined and specify a TYPE for your widget. You can get the property TYPE for any widget, but only for user defined widgets it makes sense.
Next step is to override the 2 user widget callbacks.

FUNCTION DDgui_draw_user: BYREF id$, width%, height%

Override this function to draw a user defined widget. DDgui sets the focus, so you can draw the widget as if it were on position 0,0.

FUNCTION DDgui_handle_user: BYREF id$, mx%, my%, b1%, b2%

Override this function to handle the user interaction between the user defined widget and mouse or keyboard events.
The parameters mx and my are relative to the widget's top left corner.

The callback approach might be better using CALLBYNAME. I will have to think about better ways, still.

The new "prototype" command would allow a much better integration of a user defined widget. Please post on the forum if you need your own widget and I'll extend it.

More information in the forums on www.glbasic.com

-GF