

**Designer**

<b>COLLABORATORS</b>
----------------------

	<i>TITLE :</i> Designer	
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>
WRITTEN BY		August 2, 2022
		<i>SIGNATURE</i>

<b>REVISION HISTORY</b>
-------------------------

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>Designer</b>	<b>1</b>
1.1	Designer Guide Contents	1
1.2	CopyRight	1
1.3	Introduction	2
1.4	Upgrading older versions	2
1.5	Preferences	3
1.6	Generate	3
1.7	Main Window	3
1.8	File Operations	4
1.9	Main Code	5
1.10	Using Disk Fonts	7
1.11	Open Libraries	7
1.12	Edit Window	8
1.13	Button Gadgets	9
1.14	String Gadgets	10
1.15	Integer Gadgets	11
1.16	CheckBox Gadgets	12
1.17	MX Gadgets	12
1.18	Cycle Gadgets	13
1.19	Slider Gadgets	14
1.20	Scroller Gadgets	15
1.21	Listview Gadgets	16
1.22	Palette Gadgets	16
1.23	Text Display Gadgets	17
1.24	Number display Gadgets	18
1.25	Gadget Information	18
1.26	Boolean Gadgets	19
1.27	Window code options	20
1.28	Window Sizes	21
1.29	Window IDCMP	21

---

---

1.30	Magnify Window . . . . .	22
1.31	Tags for window . . . . .	22
1.32	Text editing window . . . . .	23
1.33	Images in window . . . . .	24
1.34	Creating Bevel Boxes . . . . .	24
1.35	Editing Menus . . . . .	24
1.36	Editing Images . . . . .	25
1.37	Locale Support . . . . .	27
1.38	Credits . . . . .	27

---

# Chapter 1

# Designer

## 1.1 Designer Guide Contents

The Designer

Copyright  
Introduction  
Upgrading  
Main Window  
Preferences  
Main Code Options  
File Operations  
Libraries  
Generating  
Editing Windows  
Editing Menus  
Editing Images  
Locale Support  
Credits

(C) Ian OConnor 1994

## 1.2 CopyRight

The Designer (C) Ian OConnor 1994, All rights reserved.

---

The Designer is shareware, you may distribute copies of the demo to anybody, but the full version may not be distributed, although you can, of course, back it up if you wish as long as the backups remain at all times in your possession. This software is provided "AS IS", without warranty of any kind, either expressed or implied. The author is not responsible for any damage or loss of data due to use of this program, these are solely the users concern.

Introduction

## 1.3 Introduction

The Designer

(C) Ian OConnor 1994

Release : 1.42

This program was written to make designing Intuition interfaces for your programs easier and quicker. It will produce code to open and close

Windows  
, make  
Menus  
, make  
Images

and much more. It also has the ability to produce IDCMP handling routines for your applications along with other useful routines and if you wish will create a full program that will compile and run for the simpler windows.

It produces a file, that you can compile and use in your program, that contains all the routines you need. It is recommended that you do not edit this file because you will then be able to update it in the future for new features etc. needing only to recompile your source for a new look or extra options.

The actual production of the source is handled by a program called a producer, stored in the same directory as The Designer. You select which language you want by selecting the relevant producer in code options.

Help is provided on most functions, telling you what they do and how they are used.

Any bug fixes or updates will be released into PD in a form which will enable registered users to update their files. I will not say how often these will be released because I cannot know. It depends to a large extent upon the interest shown in this program.

Ian OConnor

## 1.4 Upgrading older versions

---

The Demo version can be used to upgrade registered users programs to the latest edition.

The extra icon on the bottom of the main window in the demo is used to create the new files required. You must use the top button in the upgrade window to select your old Designer file, it will then enable you to install the new Designer file to where you wish in full working order.

The other buttons allow the installation of other files as well, these are just copy commands. copy must be available for any of the icons to work, including the make new Designer File option.

Using this method of upgrading any registered user should be able to get the latest version without too much difficulty.

## 1.5 Preferences

This is where you can set up the options for the editor. Save ↔ writes them to disk while Use means that any changes will be lost when the computer reset.

The default producer is the one which is assumed on startup of the designer.

Main Window

Code Options

## 1.6 Generate

All code is generated from saved designer files by the producers ↔, at the moment there are two of these, Pascal and C. The generate button saves the current data ( which must have already been saved ) and then runs the selected producer on this file.

The producers can also be run from workbench or CLI with their methods of passing parameters, multiple files are supported.

Fonts

Code Options

Preferences

## 1.7 Main Window

This is the window presented on running the program, the ↔ creation of

---

windows, menus and importing of images is all handled here, as well as code production and file operations. The gadgets are as follows.

About  
: A little message.

Prefs  
: Here you can set up your own prefs for The Designer. Only options about the editor are here.

Code  
: Allows you to set code preferences changing what is produced,  
library  
options are also here.

Open, Save :  
File operations  
.

Generate  
: Saves the .des file and calls the Producer specified in the code options.

Help : Well, here we are...

New, Delete and Edit allow you to play with the  
Windows  
,  
Menus  
and

Images  
the Designer produces.

Keyboard shortcuts are underlined on the gadgets except for W, M and I which change the list displayed.

## 1.8 File Operations

Load and Save are simple and obvious, but as from V1.3 a merge ← option is available from the main window menu.

This loads in the windows, menus and images from another designer file without deleting the current data loaded. However it does not overwrite the designers code settings or the libraries that are opened, these remain as before.

All designer files are saved with a .des extension. They must be saved before they can be  
produced  
.

It is now possible to import .GUI files from GadToolsBox, this requires the GTX and nofrag libraries to be present. The result of importing a file is to get windows and menus, the assorted code options are left alone so should be set by you.



Thanks to Richard Waspe for the pascal GTX unit.

## 1.9 Main Code

Here Several options acting on the whole product are set. If  $\leftrightarrow$   
comment

code is checked then the code produced is commented to its maximum extent. This overrides the comment field of window code.

If WaitPointer is checked then a standard Release 2.0 waitpointer is included, to use it a command like this is needed :

```
SetPointer(Win, WaitPointer, 16, 16, -6, 0);
```

```
[ pWaitPointer in pascal ]
```

If IDCMP Handler is checked then the framework of an idcmp handler is produced for each window and menu designed. These functions should then be copied into your own code and edited. These are in the produced file unless you have selected make a main program file, then they are in that, see below for more info on that.

Makelibs means that

```
library
```

```
opening code will be created.
```

Make Main Program will create a dummy main called <ProjectName>main.c or .pas which can be compiled to produce a very simple program that works immediately.

This will only open the first window in the window list and open the defined libraries and making images etc. .

A basic message handler will be produced and all the functions to handle all the windows messages will be put in this main file. It will be similar to the example forms supplied. Closing the window will quit. It will not include C WorkBench startup code because I am not sure how to do that on different compilers (I do not own them), this does not affect pascal of course.

Extra parameters to the first window are not supported yet, do these yourself.

This file should only be used as a guideline for writing your main because so few programs will really be this simple.

You must make sure suitable

```
libraries
```

```
are opened for this program not
```

to crash.

As of V1.2 you now have the ability to add extra include files to the list at the beginning of the produced code. This enables you to write programs like the MultipleDemo with many copies of the same window being open at the same time. You should examine the code for the MultipleDemo carefully if you wish to do this. Most important is that you set the Window Label correctly and define the WindowNode structure properly. You do not have to use a node at all, of course, but the structure must contain all the correct fields to open the window. Then set the window to receive the suitable extra parameters and it should all work. You must also disable the definition of the window variables in the file, otherwise you will get some errors ( bottom left of window code window, at this time ).

The Code :

Pascal :

For each window 2 or 3 functions will be created :

```
Function OpenWindow'WindowLabel':Boolean;
Procedure CloseWindow'WindowLabel';
Procedure RendWindow'WindowLabel';    Optional
```

The first of these may need parameters depending on its code options. Just check the header in the unit for details.

Their also exist several global variables for each window :

```
'WindowLabel' : pWindow;
'WindowLabel'glist : pGadget;
'WindowLabel'VisualInfo : Pointer;
'WindowLabel'Gads : array[] of pgadget;    Optional
```

as well as a few others for the window gadgets.

For each menu one function is produced

```
Function MakeMenu'MenuLabel' ( VisulaInfo : Pointer): Boolean;
```

the menus should be freed with FreeMenus as normal.

The global 'MenuLabel' is a pointer to the allocated menu structure.

All images are created as const data and are allocated to chip ram by the makeimages:boolean fuction, free them on exit with free images, only free them if thy are succesfully allocated.

Several procedures are included to make life easier :

```
Procedure Settagitem( pt : ptagitem ; tag : long ; data : long);
procedure printstring(pwin:pwindow;x,y:word;s:string;f,b:byte;
                    font:ptextattr;dm:byte);
procedure stripintuimessages(mp:pmsgport;win:pwindow);
procedure closewindowsafely(win : pwindow);
function generalgadtoolsgad(kind      : long;
                            x,y,w,h,id : word;
                            ptxt      : pbyte;
                            font      : ptextattr;
                            flags     : long;
                            visinfo   : pointer;
                            pprevgad  : pgadget;
                            userdata   : pointer;
                            taglist   : ptagitem
                            ):pgadget;
function getstringfromgad(pgad:pgadget):string;
function getintegerfromgad(pgad:pgadget):long;
function GadSelected(pgad:pgadget):Boolean;
procedure gt_setsinglegadgetattr(gad:pgadget;win:pwindow;
                                tag1,tag2:long);
```

C :

For details of the functions produced read them! The definitions are in the header file produced and the functions do approximately the same as the above Pascal ones.

GTB compatability when switched on causes the CProducer to create extra code, or slightly different code as follows. This was requested and was not hard to do so here you go :

```
OpenXwindow instead of OpenWindowX
CloseXwindow as above
XNewGad instead of XNewGadgets
GD_ and GDX_ gadget references produced.
```

Alternate Includes creates slightly different C .c and .h files, having all the #includes in the .h file and having the main include .h not .c This means you do not have to recompile everything everytime.

Fonts

Generation

Locale

## 1.10 Using Disk Fonts

If the code option to make diskfonts is set then a function will be produced that opens all the fonts that the program needs, otherwise the correct fonts may not be used in the produced code when run.

Code Options

Generation

## 1.11 Open Libraries

If the procedure to open libraries is created then the libraries to open, the earliest version acceptable and whether to halt whole program if unopenable is set in the choose libraries window.

Whether to produce these functions is set in the code window. The functions created would be

```
Pascal :      Function OpenLibs:boolean;
              Procedure CloseLibs;
```

```
C          :      int OpenLibs(void);
              void CloseLibs(void);
```

Open Libs will return False in Pascal or non-zero in C if it is unable to open a library and told to fail if that library unopened. If the procedure fails then all libraries will be closed, if it does not abort on fail you should check the library you want is open before use.

Default values are set that open those libraries required by the code produced by the Designer, even if you open libraries yourself you must open these libraries :

```
Intuition  V37
Graphics   V37
GadTools   V37
```

## DiskFont V36

Your program should have a bit like this if you use these functions:

```
Pascal : If OpenLibs then
        Begin


            { rest of program }

            CloseLibs;
        End
    else
        writeln('Cannot open all libraries.');
```

```
C      : If ( OpenLibs()==0 )
        {
            /*
            Continue program
            */
            CloseLibs();
        }
    else
        {
            /*
            OpenLibs Failed
            */
        }
    }
```

Code Options

## 1.12 Edit Window

This is the main part of the program. Here you can design the windows  that will be produced for you.

The following operate on the selected or all selected Gadgets in the window at that time. To select a gadget you should just activate it by clicking on it in a way to send a message. Multiple selects are done by holding down a Shift key when selecting. Clicking on a blank bit of the window while holding down shift will create a box which will select all gadgets inside the box, if it is not cancelled with the right button.

### Gadgets

```
:
Size   : Allows you to change size of selected Gadget.
Clone  : Allows you to copy and place current selected gadgets.
Delete : Deletes selected gadgets.
Move   : Moves all selected gadgets.
Align  : Allows you to align all selected Gadgets to a given
         line and side.
Spread : Space all selected gadgets out in given direction with
```

given space in between them.

Graphics :

Bevel  
: Create and edit bevel boxes on the window.

Text  
: Create and edit text on the window.

Image  
: Place imported images on the window.

Options :

Screen : Edit edit screen mode.

Tags  
: Edit window tags.

Code  
: Edit window created code options.

Sizes  
: Edit window sizes.

IDCMP  
: Edit IDCMP message types received by program.

Help : This help text.

Other :

Magnify  
: Show window in more detail.

Code Options

Fonts

Imported Images

## 1.13 Button Gadgets

These are simple hit select gadgets with a raised bevel border.

Options :

Text           Text to place in/near gadget, not clipped.  
LabelID       Constant equal to the gadgets id produced in source.  
Place         Text location.  
Disabled      Initial state of gadget  
UnderScore   Precede a letter in Text with \_ so it is underlined.

Tags :

GA\_Disabled(BOOL)

---

Shades out gadget if true, preventing activation.

Messages :

IDCMP\_GADGETUP

IntuiMessage.IAddress contains pointer to gadget structure.

Comments :

If the gadget brings up a requester then Text should end in "...".

Gadgets

## 1.14 String Gadgets

These are Text entry gadgets with a raised ridge border.

Options :

Text	Text to place near gadget, not clipped.
LabelID	Constant equal to the gadgets id produced in source.
Place	Text location.
Disabled	Initial state of gadget
UnderScore	Precede a letter in Text with _ so it is underlined.
ReplaceMode	Gadget in replacemode instead of autoinsert mode.
ExitHelp	If help key pressed while gadget activated then message sent, see below.
TabCycle	Cycle through string/integer gadgets when tab pressed.
Immediate	Receive message when gadget selected.
Justification	Where to put the string in the gadget.
MaxChars	Maximum length of string.
EditHook	Here you are on your own. I have never experimented with this, nor do I intend too, what you type in is given directly as a tag field so make sure it is legal code. You must include the file that defines the hook function in the produced code by using the include option in the main code window.

Tags :

GA\_Disabled(BOOL)

Shades out gadget if true, preventing activation.

GTST\_String(STRPTR)

Places new string in gadget, clears if set to NULL.

Messages :

IDCMP\_GADGETUP

Received when user presses Enter, Return, Help, Tab or Shift Tab  
 if Tab then intuimessage.code = 0x09  
 if Help then intuimessage.code = 0x5F, this case should be handled carefully.

To read string

In pascal use string:=GetStringFromGad(pgadget);

In C ((struct StringInfo \* )gad->SpecialInfo)->Buffer

IntuiMessage.IAddress contains pointer to gadget structure.

Comments :

Immediate will work in all versions from 37 and up, the special case of V37 is handled properly.

Gadgets

## 1.15 Integer Gadgets

These are Number entry gadgets with a raised ridge border.

Options :

Text	Text to place near gadget, not clipped.
LabelID	Constant equal to the gadgets id produced in source.
Place	Text location.
Disabled	Initial state of gadget
UnderScore	Precede a letter in Text with <code>_</code> so it is underlined.
ReplaceMode	Gadget in replacemode instead of autoinsert mode.
ExitHelp	If help key pressed while gadget activated then message sent, see below.
TabCycle	Cycle through string/integer gadgets when tab pressed.
Immediate	Receive message when gadget selected.
Justification	Where to put the number in the gadget.
MaxChars	Maximum length of number.
EditHook	Here you are on your own. I have never experimented with this, nor do I intend too, what you type in is given directly as a tag field so make sure it is legal code. You must include the file that defines the hook function in the produced code by using the include option in the main code window.

Tags :

GA_Disabled(BOOL)	Shades out gadget if true, preventing activation.
GTIN_Number(LONG)	Places new number in gadget.

Messages :

IDCMP_GADGETUP	Received when user presses Enter, Return, Help, Tab or Shift Tab if Tab then <code>intuimessage.code = 0x09</code> if Help then <code>intuimessage.code = 0x5F</code> , this case should be handled carefully. To read string In pascal use <code>long:=GetIntegerFromGad(pgadget);</code> In C <code>((struct StringInfo *)gad-&gt;SpecialInfo)-&gt;LongInt</code> <code>IntuiMessage.IAddress</code> contains pointer to gadget structure.
----------------	--

Comments :

Immediate will work in all versions from 37 and up, then special case of V37 is handled properly.

Gadgets

## 1.16 CheckBox Gadgets

These are toggle gadgets with a raised bevel border.

### Options :

Text                   Text to place near gadget.  
 LabelID               Constant equal to the gadgets id produced in source.  
 Place                  Text location.  
 Disabled              Initial state of gadget  
 UnderScore           Precede a letter in Text with \_ so it is underlined.  
 Checked               Initial state of gadget.  
 Scale (V39)          Will allow sizing of gadget, all versions will let you  
                           change this but V39+ needed to work.

### Tags :

GA\_Disabled(BOOL)  
     Shades out gadget if true, preventing activation.  
 GTCB\_Checked(BOOL)  
     Set gadget toggle status.

### Messages :

IDCMP\_GADGETUP  
     IntuiMessage.IAddress contains pointer to gadget structure.  
     Track the state of this gadget with GFLG\_SELECTED bit in  
     gadget.Flags field.  
     In pascal use boolean:=GadSelected(pgadget)

### Comments :

The gadget structure is not synchronized with the messages, you must not rely on the state toggling each time a message is received.

Gadgets

## 1.17 MX Gadgets

These are mutually exclusive gadgets consisting of a series of ↔ buttons, only ooe of which can be active at a time.

### Options :

Text                   Text to place near gadget (V39+ only).  
 Place                  Text location (V39 only).  
 LabelID               Constant equal to the gadgets id produced in source.  
 Place                  Text location for each button.  
 Active                 Initial active button.  
 Spacing               Gap between buttons vertically, added to font height.  
 UnderScore           Precede a letter in Text with \_ so it is underlined.  
 Scale (V39)          Will allow sizing of gadget, all versions will let you



change this but V39+ needed to work.

Tags :

GTMX\_Active(LONG)  
Position to activate.

Messages :

IDCMP\_GADGETDOWN  
IntuiMessage.IAddress contains pointer to gadget structure.  
intuimessage.code contains new active option.

Comments :

Remember GADGETDOWN not GADGETUP.

Gadgets

## 1.18 Cycle Gadgets

These are mutually exclusive gadgets consisting of a series of ↔ options, only one of which can be active at a time. To select the next click on the button.

Options :

Text                   Text to place near gadget.  
LabelID               Constant equal to the gadgets id produced in source.  
Place                   Text location.  
Active                 Initial active option.  
UnderScore           Precede a letter in Text with \_ so it is underlined.  
Disabled              Initial state of gadget

Tags :

GTCY\_Labels(STRPTR\*) (set V37+)  
New null-terminated array of pointers to null-terminated strings to be used in gadgte.  
GTCY\_Active(LONG)  
Position to activate.  
GA\_Disabled(BOOL)  
Shades out gadget if true, preventing activation.

Messages :

IDCMP\_GADGETUP  
IntuiMessage.IAddress contains pointer to gadget structure.  
intuimessage.code contains new active option.

Comments :

If you implement a key for a cycle gadget remember that shift key means cycle through backwards.

Gadgets

## 1.19 Slider Gadgets

These are proportional gadgets that allow you to select a number in a range. ↔

### Options :

Text	Text to place near gadget.
LabelID	Constant equal to the gadgets id produced in source.
Place	Text location.
Min Level	Lowest point possible.
Max Level	Highest point possible.
Level	Initial level.
Freedom	Whether to move horizontally or vertically.
Immediate	Whether to receive a message on gadget activation.
Relverify	Whether to receive a message when gadget released.
Disabled	Initial state of gadget.
Display	Print level by gadget.
UnderScore	Precede a letter in Text with _ so it is underlined.
Level Place	Where to print level if printed by gadget.
Level Format	C String format for level printed.
Max Level Len	Maximum length of string printed.
DispFunc	Here, you are on your own. I have never experimented with this, nor do I intend too, what you type in is given directly as a tag field so make sure it is legal code. You must include the file that defines the function in the produced code by using the include option in the main code window. it should be something like this (LONG(*function)(struct Gadget *,WORD))

### Tags :

GTSL_Min(WORD)	Minimum level.
GTSL_Max(WORD)	Maximum level.
GTSL_Level(WORD)	Change current level.
GA_Disabled(BOOL)	Shades out gadget if true, preventing activation.

### Messages :

IDCMP_GADGETUP	User Finished adjusting slider. IntuiMessage.IAddress contains pointer to gadget structure. intuimessage.code contains new level.
IDCMP_GADGETDOWN	User begins to adjust level.
IDCMP_MOUSEMOVE	If level changes then intuimessage.code contains new level.

### Comments :

If you are working with negative levels then make sure you typecast into words properly as code field of messages is UWORD.

## Gadgets

**1.20 Scroller Gadgets**

These are proportional gadgets that allow you to select a region  $\leftrightarrow$  in a range.

## Options :

Text	Text to place near gadget.
LabelID	Constant equal to the gadgets id produced in source.
Place	Text location.
Top	Highest point possible.
Total	size of region.
Visible	Amount of range visible.
Immediate	Whether to receive a message on gadget activation.
Relverify	Whether to receive a message when gadget released.
Disabled	Initial state of gadget.
Arrows	Include Arrows on end of bar.
UnderScore	Precede a letter in Text with <code>_</code> so it is underlined.
Freedom	Whether to move horizontally or vertically.
Arrows	Size of arrows in screen pixels.

## Tags :

GTSC_Top(WORD)	Maximum level.
GTSC_Total(WORD)	Size of region.
GTSC_Visible(WORD)	Amount in selected part of region.
GA_Disabled(BOOL)	Shades out gadget if true, preventing activation.

## Messages :

IDCMP_GADGETUP	User Finished adjusting slider. IntuiMessage.IAddress contains pointer to gadget structure. intuimessage.code contains new level.
IDCMP_GADGETDOWN	User begins to adjust level.
IDCMP_MOUSEMOVE	If level changes then intuimessage.code contains new level.

## Comments :

If you are working with negative levels then make sure you typecast into words properly as code field of messages is UWORD.

## Gadgets

## 1.21 Listview Gadgets

These gadgets provide a way of displaying a list.

### Options :

Text	Text to place near gadget.
LabelID	Constant equal to the gadgets id produced in source.
Place	Text location.
Active	Initial active option.
Top	Initial top of list position.
Spacing	Space between each item.
Scrollwidth	Width of scrollbar.
UnderScore	Precede a letter in Text with _ so it is underlined.
ReadOnly	Make Gadget non selectable.
CreateList	Make List of items as seen on screen.
Display	Display Selected item.
Join,Split	Connect/Disconnect a string gadget to the listview, this enables easy editing of the items.
CallBack	(V39) Here you are on your own. I have never experimented with this, nor do I intend too, what you type in is given directly as a tag field so make sure it is legal code. You must include the file that defines the function in the produced code by using the include option in the main code window.

### Tags :

GTLV_Labels(struct List*)	List to put in listview.
GTLV_Top(UWORD)	Topmost displayed item.
GTLV_Selected(UWORD)	Set selected item.
GTLV_MakeVisible=GT_TagBase+78 (LONG) (V39)	Make item visible.
GA_Disabled(BOOL) (V39+)	Shades out gadget if true, preventing activation.

### Messages :

IDCMP_GADGETUP	IntuiMessage.IAddress contains pointer to gadget structure. intuimessage.code contains new selected item.
----------------	--

### Comments :

If you implement a key for a cycle gadget remember that shift key means cycle through backwards.

Gadgets

## 1.22 Palette Gadgets

These gadgets provide a way of selecting colours.

## Options :

Text	Text to place near gadget.
LabelID	Constant equal to the gadgets id produced in source.
Place	Text location.
Depth	Depth of palette requester, 0 for screen (Designer feature, not gadtools). it will also mean a variable <WinLabel>Depth will contain this depth (not the number of colours).
Color	Initial Colour selected.
Color Offset	Start colour from screen.
Disabled	Initial state of gadget.
UnderScore	Precede a letter in Text with _ so it is underlined.
Indicator Left	Place indicator to left.
Indicator Top	Place indicator to top, use either of these for V39 indicator.
Indicator size	Size of indicator, set 20 if program only V39, so will work on V37.

## Tags :

GTPA_Color(WORD)	Set selected colour.
GA_Disabled(BOOL) (V39+)	Shades out gadget if true, preventing activation.

## Messages :

IDCMP_GADGETUP	IntuiMessage.IAddress contains pointer to gadget structure. intuimessage.code contains new selected colour.
----------------	--

## Comments :

If you implement a key for a palette gadget remember that shift key means cycle through backwards.

## Gadgets

## 1.23 Text Display Gadgets

These gadgets just display text, they send no messages.

## Options :

Text	Text to place near gadget.
LabelID	Constant equal to the gadgets id produced in source.
Place	Text location.
Bevel	Draw a bevel box around gadget.
CopyText	Copy string passed so can delete it, only applies to first text.
Display Text	First text to display.
V39	Set true to use following.
Frontpen	Text colour.
Backpen	Text background colour.
Justification	Where to put text.
Clip	Whether to clip at borders.

Tags :

```
GTTX_Text (STRPTR)
    Put new text in window.
```

Comments :

Fiddle around with clip and justification in V39 to get different results, I think its safe to do.

Gadgets

## 1.24 Number display Gadgets

These gadgets just display numbers, they send no messages.

Options :

Text	Text to place near gadget.
LabelID	Constant equal to the gadgets id produced in source.
Place	Text location.
Bevel	Draw a bevel box around gadget.
Number	First number to display.
V39	Set true to use following.
Frontpen	Text colour.
Backpen	Text background colour.
Justification	Where to put text.
Clip	Whether to clip at borders.
Number Format	C String controlling number format, empty gad for none.
Max Num Len	Supposed to limit string length, not sure if it works.

Tags :

```
GTNM_Number (LONG)
    Put new number in gadget.
```

Comments :

Fiddle around with clip and justification in V39 to get different results, I think its safe to do.

Gadgets

## 1.25 Gadget Information

Note :

If you intend to use the V39 tags that some gadgets have then you should test the program in V37, if applicable, to make sure you do not get different results.

For example if you scale the checkboxes to a different size then they will look rather different under different OS2 and OS3. The same with MX, Text and Number kinds is true, as well as some small changes to

others.

All modifiable tags are detailed in the gadget information sections.

The procedure `GT_SetSingleGadgetAttr` is supplied in any produced pascal source so that you can easily change tag values with only one call.

Gadgets :

- Button
- String
- Integer
- CheckBox
- MX
- Cycle
- Slider
- Scroller
- Listview
- Palette
- Text
- Number
- Boolean
- Edit Window

## 1.26 Boolean Gadgets

These are constructed on top of the GadTools Generic class, `boolean` ←

gadgets are those used in buttons, toggle switches, mutual excludes and so on. The inclusion of this type is meant to allow the use of some gadgets with definable imagery. You can choose the placing and type of text with much more precision, select the activation methods, the highlighting method and images to use in the different state and the initial state. Experimentation will show what can be done.

`OnGadget` and `OffGadget` should be used to enable/disable and the messages received will be `IDCMP_GADGETUP` and `IDCMP_GADGETDOWN` if you select `IMMEDIATE` and `RELVERIFY` respectively. The toggle gadgets in the Tools window are of this type and `GetFile` gadgets can be made using this type.

You should still use  
images

---

that look like the other types. The style of gadgets should be kept. To make the gadgets work properly you should set their size to be the same as the images used.

Edit Window

## 1.27 Window code options

These options change the kind of procedures produced to open and close the edit window

You should really check if the window is open unless it is only called once, then this would be wasted code.

Opening only if can create gadgets is also a good idea. Having more than one gadget font can make windows look over-complicated and creates larger programs, but is sometimes required.

Return boolean can allow a program to fail if the window is unopenable, this only applies to pascal because you can ignore the return in C.

A custom message port can be supplied and the window will be closed safely.

Calculating border sizes allows for different height of title bars, not doing so fixes them at the edited size.

Producing a pwidget array allows referencing the gadgets by your code and is usually necessary.

Attaching a menu created is easily done and options allow definition of the code produced.

Commenting code allows you to read over the unit and see what is happening, commenting is done if the Main Code options Comment is set.

WorkBench AppWindows allow icons to be dragged onto your window, if it is on the Workbench screen. It requires a separate message port which is supplied as a parameter to the openwindow procedure, also supplied to the openwindow function is a long for the appwin id.

The gadget list window, which can be opened from here, or automatically via preferences, allows you to change the order of the gadgets. This changes their gadget IDs the first gadget ID is also set here, usually 0. This window can also be used to edit gadgets which cannot be activated, ie if you put them behind another accidentally or an error occurs and they cannot be created. High acts in the same way as clicking on the window does, ie Shift High does not unhighlight other gadgets.

Setting scale using screen font makes all gadgets and bevel boxes change size so that, in theory, the window looks the same and any larger screen font can be accommodated. It is also possible to set Texts on the screen to use the screen font so everything looks the same. Everything is laid out properly I hope and the window size is changed. To make sure this options works OK for your window you should test it in the Designer with several different fonts and sizes. Proportional fonts seem to work OK most of the time but there are probably exceptions.

The params and " do not define some pointers " should be used in the same way as the MultipleDemo shows, do not experiment with these values as they will stop your code working. If these options are used then the same designer file will no longer produce both C and Pascal source



that works, as all the demos other than MultipleDemo do. I would suggest you base all your multiple window code around the shell of the MultipleDemo unless you really know what you are doing, and what the Producers make. The structure of the demo is not dissimilar to that of the Designer itself, with many different types of nodes and only one message port, this way most things can be done at the same time, eg edit a window and a menu together, although it can be quite hard to keep everything up to date with everything else. You delete an Image and the Designer has to check every menu, item, subitem, window ,boolean gadget and window image, then it must check which edit windows need updating or closing, a long job.

SuperBitmap support allows you to create a bitmap and pass it to the openwindow function or allow the produced code to create its own bitmap. You should always set GimmeZeorZero on a superbitmap window. The bitmap created by the produced code will be the same size as the windows maximum size, so you should reduce this to the minimum for memory reasons. If you do not you will always end up with a 1200 by 1200 bitmap which needs loads of chip ram.

Localisation of gadget texts, window titles, screen titles and window texts are now supported, see

locale  
for more info.

## 1.28 Window Sizes

Allows you to directly edit the  
window  
size, zoom size and the maximum

and minimum sizes. All changes will be made to the window when OK or Update are selected but if you move or size the window before updating then your input will be overwritten with the new size. These sizes are the actual ones on screen, including the borders, if border sizes are calculated then the window size will be modified suitably. If InnerWidth and InnerHeight are not set to 0 they will be used instead of width and height. It would probably be sensible to use InnerW and InnerH all the time, this along with calculating border sizes will produce windows as good as a gimmezz, as far as sizeing goes. When InnerWidth and InnerHeight are in use the width and height values are not editable.

Window sizes can be scaled for different screen fonts - see window code

## 1.29 Window IDCMP

Choose which IDCMP messages will be sent to the  
edit window  
by Intuition.

See RKM for full documentation. Suitable IDCMP will be added for gadgets as used by the window anyway.

IDCMP Flags :

MOUSEBUTTONS : Supply info about mouse button presses.

```

MOUSEMOVE      : Tell when mouse moves.
DELTAMOVE      : As above with change of position.
GADGETDOWN     : Gadget message.
GADGETUP       : Gadget message.
CLOSEWINDOW    : CloseWindow gadget pressed.
MENUPIK        : Menu Item Selected.
MENUVERIFY     : Is it OK to draw a menu ?
MENUHELP       : Help key pressed on menu item.
REQSET         : Requester set on window.
REQCLEAR       : Requester removed from window.
NEWSIZE        : Window has been resized.
REFRESHWINDOW  : Window needs redrawing.
SIZEVERIFY     : Can window be resized ?
ACTIVEWINDOW   : Window made active.
INACTIVEWINDOW : Window deactivated.
VANILLAKEY     : Vanilla key code passed.
RAWKEY         : Raw key code passed.
NEWPREFS       : 1.3 Prefs changed.
DISKINSERTED   : Floppy disk inserted.
DISKREMOVED    : Floppy disk removed.
INTUITICKS     : Timing message.
IDCMPUPDATE    : Boopsi Message.
CHANGEWINDOW   : Window Sized or moved.

```

### 1.30 Magnify Window

Allows you too see what you are doing in more detail on a screen around the mouse pointer. A gimmick but can be useful on a superhires-interlace screen or similar.

Sometimes it overwrites the windows borders when it is sized, not quite sure how to stop this, although it seems to be perfectly safe.

Its probably a good idea to keep it quite small, otherwise it slows everything down rather a lot.

A complemented dot shows where the mouse pointer actually is.

### 1.31 Tags for window

The tags specified here define a lot of details for your edit window

. Not

all will be used while editing but they will all be in the code generated.

Specific Information

```

WindowTitle    : Title string for window.
ScreenTitle    : Title string for screen when window is active.
WindowLabel    : Label referred to in source.
CustomScreen   : Allows Custom Screen Pointer to be passed to window

```

opening routine.

PubScreen : Similar to above but Public screen.

PubScreenName : Pass a pointer to a null terminated string giving name of public screen to open on.

PubScrFallBack : Fall back to default screen if cannot find public requested.

MouseQueue : Mouse message backlog limit.

RptQueue : Repeat key backlog limit.

SizeGadget : Do you want a sizing gadget ?.

SizeBRight : Put Size Gadget in right border.

SizeBBottom : Put Size Gadget in bottom border.

DragBar : Allows window title bar dragging.

DepthGadget : Allows user to change window depth.

CloseGadget : Window has a close gadget.

ReportMouse : Send mouse movements to window.

NoCareRefresh : Do not receive refreshwindow messages, bad idea with gadtools.

Borderless : Make window borderless, usually just backdrop windows have this.

Backdrop : Window is always at the back, can only have one per screen.

GimmeZeroZero : 0,0 of window is below title bar and right of left border.

Activate : Activate window on opening.

RMBTrap : Trap menu events, do not allow menu selections.

SimpleRefresh : No intuition refreshing at all.

Smartrefresh : Intuition handles most refreshing.

Autoadjust : Move/Size window so that it goes on the screen.

MenuHelp : Receive IDCMP\_MENHELP when user presses help button on menus.

Zoom : Supply zoom gadget array of values.

NewLookMenus : In V39 this will make windows use the new standard. This should be left true. All Designer produced menus are newlook from Designer V1.3 and if you set this to false then strange results may be produced, this effects only V39.

NotifyDepth : IDCMP\_CHANGEWINDOW messages with code = WCODE\_DEPTH will be sent when windows depth is changed (V39).

TabletMessages : Receive graphics tablet input (V39).

You can use any of the above V39 tags in your programs to compile with V37 includes, and run on V37 machines.

For full information see manuals.

## 1.32 Text editing window

Editing strings to be placed in the window  
 , it is all pretty self

explanatory. All fonts are supported and can be easily selected. The drawmodes are standard as well, just try them if you are not sure what they do.

The text gets displayed at the bottom of the window, Update puts the texts

on the edit window, if placed.

All the texts must be placed before they are drawn. Clicking on the edit window in edit text mode allows you to move the currently selected text.

Setting use screen fonts enables a standard look in a window using scaled gadgets.

### 1.33 Images in window

Any image loaded in can be placed on the edit window

. They are removed

if the image is deleted. A list of those placed is available, an image can be placed any number of times on a window.

The exact positioning of an image can be changed by changing the numbers on the image choosing window. The image drawing gadget works in the same way as the

text

drawing gadget, it moves the currently selected image about the window.

### 1.34 Creating Bevel Boxes

These use the GadTools BevelBox procedure to draw 3-D Bevel Boxes on

the

edit window

. Normal boxes bring out an area to show it can be selected, Recessed boxes show the user it cannot be selected and Double boxes separate out areas of a window.

Bevel Boxes cannot be selected on screen so you have to edit them using the options in the edit window. Update redraws the edit window so that you see any changes you have made to box types.

If a scaled window is selected these will be resized accordingly.

If you use the V39 boxex you will get a normal box on a V37 machine.

### 1.35 Editing Menus

Menus can be created as stand alone to be used as you wish, or they can

be attached to windows designed in the program. The layout of the menus is all pretty obvious to an amiga user. Titles are the left column, Items in the centre and SubItems to the right.

The

font

and colour of the text can be changed easily, and graphic items can be used instead of text, the second listview in each column contains a list of all imported

images

.

There is a problem with these if you try to use an image taller than the screen, the machine crashes, or at least, mine does.

The menu you create can be tested using the Test button, this updates the menu attached to the menu edit window. This is not necessary if the Autotest option is set in

prefs

. The option to turn autotesting off

exists because it can slow down menu creation quite a lot.

There must be at least one Title on each menu, the number of Titles, Items and SubItems is limited only by intuition.

Mutual exclusion is possible for items and subitems. The items/subitems you wish to exclude from the selected item/subitem should be checked on the menu. You should 'Test' the menu before doing this if it is not 'auto tested' to make sure it is up to date. Failure to do this might cause problems reading the menus. I recommend you set the checked bits of all items to be excluded while excluding them to make the job easier, turning then off afterwards to get the required menu actions. You must test the menu for this to take effect, it does not work otherwise. If it autotests then it is impossible to set up most situations.

If the code option IDCMP Handlers is set in the

code window

then a

procedure will be produced for each menu which is the framework for processing input for the menu. You should copy these procedures into your own program and edit them so they carry out the required actions. If you want MENUHELP then copying this procedure twice will enable response to those messages also.

As of Designer V1.3 all menus are produced with the NewLookMenus option. This will only affect programs when running under OS3.0 and up. It will make the menus look like the standard WB3 menus. The windows need to have the WA\_NewLookMenus tag set to true and that is now the default for the Designer windows. This has no effect with earlier OSs.

## 1.36 Editing Images

Any non-Ham IFF image can be imported into the Designer and the ←  
code

produced will contain an Image structure which can be used as desired by you. Most of the fields in this image structure are defined by the image itself but you can change the PlanePick and PlaneOnOff fields.

The PlanePick field specifies which bitplanes the image is drawn in. For each bitplane in the image there must be a corresponding destination bitplane in PlanePick. The designer will ensure that the PlanePick value is always legal.

The PlaneOnOff field just selects whether the planes not written to by the image are set or cleared. Default is all cleared.

Use the view button to update the display so you can see what the image looks like.

To move the images into chip ram this is necessary :

```
Pascal : If MakeImages then
        Begin
```

```

        { rest of program }

        FreeImages;
    End
else
    writeln('Cannot make images.');
```

C : It is only necessary to call this function if your compiler does not support `__chip`. Set the option in the main code

window to choose whether `__chip` is used or these functions ←  
are produced.

```

If ( MakeImages()==0 )
{
/*
Continue program
*/
FreeImages();
}
else
{
/*
MakeImages Failed
*/
}
```

Colour maps are created in the produced files and can be used when an image display window is opened, set whether they are or not in prefs

The maps are only produced if the images imported have a colour map, it is not required for success. At the moment only 4096 colours are supported, 24 bit palettes are converted down to 12 bit internally. `LoadRGB4` is used to set these to a viewport. To set a colourmap to a screen use :

```

Pascal      : LoadRGB4( @pscr^.viewport, pword(colours), numcolours);
C           : LoadRGB4( &Scr->ViewPort, (UWORD *)colours, numcolours);
```

If you wish to edit a window with an imported palette then the only way to do this is at the moment is to open an image view window on the edit screen.

The imported images can be used in  
windows  
,  
boolean gadgets  
and  
menus  
in

the designer.

Images can now be replaced, this allows you to change or update an image without having to specify all the places it is used, you used to have to load a new image, delete an old one and then go through putting it where it belongs. Just select this from the menu.

Warning : If you replace an image used in a menu with one too tall for the screen it crashes my computer, so replacing should be done carefully,

also see menu help about this.

## 1.37 Locale Support

It is now possible for you to support Locale in your programs which are made with The Designer. Menus and Window strings are supported allowing you to produce code which has every string localized. You can also add your own strings to those to be put in the .cd file. The producers will create a .cd file if the option in code options is set, this will then allow you to create a catalog file with catcomp or similar.

It is necessary for you to have a program like this to create .catalog files but these are not necessary for the program to run, internal defaults will be used if either of the catalog or locale.library is unavailable.

To use the catalogs in your code you must open the catalog and close it when done. The functions to do this are `Open+basename+Catalog(NULL,NULL);` and `Close+basename+Catalog();`

In Pascal code is produced for the locale.library functions as their is no unit for these in V1.1, Version 3 includes are necessary for C.

You should create most of your program and make sure it all works properly before creating any catalog files, as these must be up to date or problems will occur. You should definately increase the locale version number each time you recreate the catalogs and translations.

For full instructions on how to process .cd and .ct files see docs on catcomp or flexcat.

Make sure you do not mix up old and new version of the catalog files.

When you create new Windows and Menus the locale options are set depending on the preferences option : Localize Everything.

## 1.38 Credits

I wish to thank the following for work that has helped me write the code for these programs.

HiSoft, D-House and Christen Fihl  
for the HSPascal compiler.

Matt Dillon  
for DICE 2.07.56R

Jan van den Baard  
for the GTX.library.

Richard Waspe  
for the pascal GTX unit.

Jochen Wiedmann  
for flexcat V1.0 ( Available on Aminet )

C= ( The development teams )  
for the Amiga.

---