Designer

COLLABORATORS			
	TITLE : Designer		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		September 19, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Designer **1**

1.1 Designer Guide Contents . . . 1
1.2 CopyRight . . . 2
1.3 Introduction . . . 2
1.4 Upgrading older versions . . . 3
1.5 Preferences . . . 3
1.6 Tutorial . . . 4
1.7 Main Window . . . 5
1.8 File Operations . . . 6
1.9 Main Code . . . 6
1.10 Using Disk Fonts . . . 8
1.11 Open Libraries . . . 9
1.12 Edit Window . . . 10
1.13 Button Gadgets . . . 12
1.14 String Gadgets . . . 12
1.15 Integer Gadgets . . . 13
1.16 CheckBox Gadgets . . . 14
1.17 MX Gadgets . . . 15
1.18 Cycle Gadgets . . . 15
1.19 Slider Gadgets . . . 16
1.20 Scroller Gadgets . . . 17
1.21 Listview Gadgets . . . 18
1.22 Palette Gadgets . . . 19
1.23 Text Display Gadgets . . . 20
1.24 Number display Gadgets . . . 20
1.25 Gadget Information . . . 21
1.26 Boolean Gadgets . . . 22
1.27 BOOPSI Objects . . . 23
1.28 Window code options . . . 27
1.29 Window Sizes . . . 30

1.30 Window IDCMP 30
1.31 Magnify Window 31
1.32 Tags for window 31
1.33 Text editing window 32
1.34 Images in window 33
1.35 Creating Bevel Boxes 33
1.36 Editing Menus 33
1.37 Editing Images 35
1.38 Locale Support 36
1.39 Screen Information 37
1.40 Credits 40
1.41 Producers 41
1.42 AsmProducer 41
1.43 CProducer 43
1.44 PasProducer 44
1.45 Demonstration Programs 46

Chapter 1

Designer

1.1 Designer Guide Contents

The Designer

Copyright

Editing Windows

Introduction

Editing Menus

Upgrading

Editing Images

Tutorial

Editing Screens

Main Window

Locale Support

Preferences

Producers

Main Code Options

Credits

File Operations

Libraries

Demonstration Programs

1.2 CopyRight

Introduction

1.3 Introduction

The Designer

(C) Ian OConnor 1994

Release : 1.50

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

Windows
,
Menus
,
Screens
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, this 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.

If you have any comments, suggestions, or bug reports then you can contact me at the given postal address or email at

ijo1000@cus.cam.ac.uk

this address will not exist beyond June 1995.

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 update the file as required. You must use the file requester to select your old Designer file, it will the enable you to transform the demo Designer file to a fully working version.

You should then use the Install script to copy the required files to where you wish.

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

It should not be necessary to delete old Designer files, the new files will overwrite them.

1.5 Preferences

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

```
Default Producer      : This is the
              Producer
               which is assumed on startup
                          of the Designer.
Open About When Run   : Open about window on startup.
Delete Window AYS     : Confirm Designer window deletion.
Delete Menu AYS       : Confirm Designer menu deletion.
Delete Image AYS      : Confirm Designer image deletion.
Delete Gadget AYS     : Confirm Designer gadget deletion.
Make Icons            : Crate Icons for saved Designer files.
Load AYS              : Confirm loss of current data.
BackDrop Tools Win    : Allows tools window for editing windows to
                          become dragable and depth arrangable.
Auto Open Gadget List : Open gadget list window when edit window screen
                          opened.
Images with palette   : When displaying images set screen to use
```

```
                          the images palette.
 Create File Backups   : When saving files rename previous saved version
                          to Name.Bak, use load to recover.
 Auto test Menus       : See menu help.
 Revert AYS            : Confirm revert when selected.
 Localize everything   : Default value of localization in windows and
                          menus.
 Old Style error msgs  : Error reports in title bar instead of requesters,
                          under really low memory conditions requesters
                          can fail while old style error message will not.
 Delete Screen AYS     : Confirm Designer screen deletion.
 Screen Editor on scr  : Open screen edit window on created screen, if
                          possible to do so.

 The Designer saves its data in the proper way in the env: and envarc:
drawers in a subdirectory called Designer.


                Main Window

                Code Options
```

1.6 Tutorial

```
                 This section will run through the creation of a simple program ↩
                    with The
Designer. You will create a simple window, put a few gadgets on it,
compile it and run it.

 First run The Designer and press "New" to create a new window. This will
make a window called "New Window 0" and add itto the list in the main
window. Select this window from the list and click on "Edit".
 You will be presented with a new screen containing an emty window and a
tools window. This is where you must design your GUI.
 To create a gadget select the type you wish, Cycle for example, and move
over to the empty window. Click in the window and holding the LMB down
drag the box into the shape of the gadget you want. The RMB will cancel
this operation, while relasing the LMB will add the gadget to the window
and open a Gadget Edit Window. Here you can modify the gadgets details,
click on OK when you are done. Press help for help on that gadget kind.
 Adding texts, bevel boxes and images is done similarly.
 Once you have created a few gadgets etc. click on the window close gadget
or select exit from the menu. You must now modify the code options for
this file. Click on "Code" and select the Producer you want. Also set the
following CheckBoxes to ticked :
     MakeLibs
     Make Main Program
 If you are using HSPascal V1.2 then you must always set HSPascal 3.1
units.
 Close the code window and save the Designer file as test.des to RAM:
Once you have saved the Designer file you can click on "Generate" and
the Producer should be run. If you have not got enough memory you may
have to quit and run the Producer without the Designer in memory.
 If you now look at RAM: you should find the source files you require
```

to create a fully working program. To see how to compile this code see
the Producers details.

AsmProducer

CProducer

PasProducer

1.7 Main Window

This is the window presented on running the program, the creation of
windows, menus and the importing of images are handled here, as well as
code production and file operations.

Gadgets operations:

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 and
Produces
the loaded data.

Help : Well, here we are...

New, Delete and Edit allow you to play with the
Windows
,
Menus
,
Images
and
Screens
the Designer produces.

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

Menu operations (where different from gadgets) :

```
  Clear All  : Delete all Windows, Menus and Images.
  Merge      : Loads the selected file, but does not delete the current
               Windows, Menus and Images. The currently loaded code
               options are kept.
  Save       : Saves without the file requester, data needs to have been
               saved already
  Save As    : As Save gadget.
  Revert     : Load last saved version over current data, effectively lose
               modifications.
  Import GTB : Allows you to select a GadToolsBox file to be imported so
               you can edit it with the Designer. The Source produced is
               not identical and minimal conversion of any program using
               this code may be required.
  Quit       : Quit

  @{ " File Operations " link file}
```

1.8 File Operations

```
                The Merge option loads in the windows, menus, images and screens ↩
                    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 the merge.

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

 It is 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. To merge a Designer fileand a GTX file you should
import the GTX file and save it to T: then load the Designer file and
merge the file on T:.
 Thanks to Richard Waspe for the pascal GTX unit.
```

1.9 Main Code

```
                Here Several options acting on the whole product are set.

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

 Make WaitPointer Data
   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 ]

Create IDCMP Handler
  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 this.

Make Library Code
  This means that
              library
               opening code will be created.

OpenDiskFonts
  Create function to open disk fonts as required.

Make Main Program
  This will create a dummy main source file called <ProjectName>main.c,
 .pas or .s 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. .

Open first screen
  You can choose to open a screen in the main program, this will be the
 first of the screen list. If the first window opens on a custom screen
 it will open on this screen, see {"ScreenDemo" link demos}.

  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 or Asm 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, 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.

Produce Locale .cd
Produce Locale .ct
  These will cause .cd and .ct files to be created which can be used
 with suitable programs to create catalog files for different languages.
  See
              locale
               for more info.

HSPascal 3.1 units
  Version 1.2 of HSPascal is now available and requires slightly
 different code, to get this set this flag.

GTB Compatability
```

```
   Set this and the C code produced will be slightly different and is
  more compatible with code produced by GTB. See the
               CProducer
                text
  for more information.

 Use __chip in C
   This makes c put the image data of imported images in chip ram so it
  is not necessary to use MakeImages or FreeImages.

 Alternate Includes
   The arrangement of the C include files in the main program are
  altered slightly to allow partial compilation each time you compile
  your data.

 Include
   As of V1.2 you have the ability to add extra include files to the list
  at the beginning of the produced code. The extra filenames should be
  sepearated by commas.
   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 ).
   These options are not supported in the AsmProducer, instead I chose to
  implement them differently by copying the result of OpenWindow, see the
  MultipleDemo source for more information. This method is also possible
  in the other Producers.


               AsmProducer

               CProducer

               PasProducer

               Fonts

               Producers

               Locale

               Libraries
```

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

Producers

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

```
Asm    :        OpenLibs
                CloseLibs

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

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

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.
In assembly d0 will contain 1 if OpenLibs fails, otherwise 0.
CloseLibs sets no return but the contents of d0 will be destroyed, see
commented code for more details.
If the procedure fails then all libraries will be closed, if it does not
abort on fail for some particular library then you should check the
library you want is open before using it.
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
```

dos.library is always opened in assembler if OpenLibs is created, if this
is unopenable then OpenLibs fails.

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

```
Asm     :       jsr     OpenLibs
                tst.l   d0
```

```
                bne     LibsFailed

                ..
                ..

                jsr     CloseLibs
            LibsFailed:

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

Pascal  :   If OpenLibs then
              Begin

                { rest of program }

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



               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.


              BOOPSI Objects

              Gadgets
                :
```

```
          Move   : Moves all selected gadgets.
          Align  : Allows you to align all selected Gadgets to a given
                   line and side.
          Size   : Allows you to change size of selected Gadget.
          Delete : Deletes selected gadgets.
          Clone  : Allows you to copy and place current selected gadgets.
          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 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    : Its me again....

 Menus (Where different from above) :

          UpdateWin  : Redraw everything.

               Magnify
                  : Show window in more detail.
          ScreenFont : Allows changing of screen font.
          Exit       : Finish editing window

          Gadget List    : Open Gadget List Window.
          Highlight All  : Set all gadgets to selected.
          Highlight None : Deselect all gadgets.
          Edit High Gads : Open edit windows for all highlighted gadgets.


                Code Options

                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, reorder in gadget list window.
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 _ 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,
                     reorder using gadget list window.
     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 intuimessage.code = 0x09
         if Help then intuimessage.code = 0x5F, this case should be
           handled carefully.
         To read string
           In pascal use  long:=GetIntegerFromGad(pgadget);
           In C ((struct StringInfo * )gad->SpecialInfo)->LongInt
         IntuiMessage.IAddress 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 ooe 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,
        IntuiMessage.IAddress contains pointer to gadget structure.

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 ↩
                   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 _ 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,
         IntuiMessage.IAddress contains pointer to gadget structure.

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_SetSingleGagdetAttr 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
```

```
                Objects

                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.

Options:
     LabelID        Constant equal to the gadgets id produced in source.
     Width          Gadget width.
     Height         Gadget Height.
     Text           Text to place near gadget.
     X              Position of text.
     Y              Position of text.
     DrawMode       Mode of text.
     INVERSVID      Inverse text.
     Use Text       Put text in gadget?
     FrontPen       Foreground colour.
     BackPen        Background colour.
     ToggleSelect   Make toggle gadget.
     Immediate      To receive GADGETDOWN messages.
     Relverify      To receive GADGETUP messages.
     FollowMouse    Send mousemove messages while active.
     Selected       Initial toggle state.
     Disabled       Initial disabled State.
     GadgetRender   UnSelected image.
     SelectRender   Selected image.
     GADGHNONE      No highlighting.
     GADGHCOMP      Complement when selected.
     GADGHBOX       Complement Box when selected.
     GADGHIMAGE     Alternate image for highlighting.

Tags:
     None

Messages:
      IDCMP_GADGETUP
          Gadget released.
      IDCMP_GADGETDOWN
          Gadget pressed.

Comments:
     Experimentation will show what can be done.
     OnGadget and OffGadget should be used to enable/disable.
     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.


                Gadgets
```

1.27 BOOPSI Objects

```
 You should now be able to include any BOOPSI object in the Designers
Produced code, all public objects should be creatable in the Designer.
 Many kinds of tags are included in the editor and any kind can be set
using extra include files.
 These things can crash anything with ease. Make sure you have nothing
important in memory before experimenting with these. This is not the
Designers fault, it just does what you tell it. Especially be careful
with objects that free other objects and disposal of them. Freeing
memory twice is not a good practice.
 I recommend you only edit things that you have full documentation for,
I cannot ducument even the simple types here due to the amount of
information available about them and the details required to make them
work. For the best information see the Rom Kernal Manuals or AutoDocs.

Highlighting :
  It is unfortunately not possible to highlight objects by clicking on
 them at the moment. You can hold down shift and drag a box over the
 corner of the object to highlight it, or you can use the GadgetList
 window.
  I now prefer to keep the GadgetList window open always, double click
 on the gadgets name in the listview to edit it.
  The highlighted rectangle does not actually have to have anything to
 do with the objects position.

Refreshing :
  If you use image objects then they will be redrawn by rendwindow when
 the window is refreshed. These might draw over the objects so it may be
 that GT_RefreshWindow and RefreshGList need to be called after window
 refreshing. If this is not neccessary then it should not be done because
 it slows down window refreshing a lot.

Creation :
  All objects are created after all gadtools gadgets, regardless of
 gadget list order. Relative object order and Relative gadtools gadget
 order are maintained however. Any gadget with a tag set as another
 object after it in the list is set using SetAttrs or SetGadgetAttrs.

Borders :
  I beleive it is possible to create gadget objects in the borders of
 windows. This will severely mess up the Designers offset handling if
 you do this in the left or top borders. Do so at your own risk and
 do not expect succesful handling.
```

It should also be noted that the Edit Window in the editor is created before the objects, while the opposite is true of produced code so the borders will be handled differently in the editor and the produced code, expect more strange results.
This should trouble very few people, I hope.

The edit window :

Gadget Label
Reference number.

Class Name
If the object is of class type is public then this string will be used in NewObject to create the object, if the class type is private then this is a pointer to the class, defined in a file included by the produced code, in assembly the contents of this address is copied using move.l . This enables you to use custom class in the produced code, although these will not be creatable in the editor.

Class Type
See description of Class Name.

Object Type

Gadget
If defined as a gadget the object will receive a GA_Previous tag on creation, so do not pass another of these. It will also be used as the next previous gadget.
SetGadgetAttrs will be used on this instead of SetAttrs.

Image
If defined as an image then the function DrawImageState will be used to render the image in the window render function.

Other
The object will be created and forgotten by the Designer unless linking to another object is required. No rendering of GA_Previous linking is done.
You may need to set some images to this if they are linked to other objects and should not be rendered.
You may also need to set some gadgets to this if you do not wish them to be placed in the gadget list, if they are in a group, for example.

Create
This only applies inside the editor, if you wish the Designer to attempt to create this object then set to true, otherwise a button of set minimum size will be created.

Dispose
Set this and the Designer and the code produced will try to dispose of the object, this may not be desirable, for example if the object is a child object of another object and is freed by its parent automatically.

Scale
Do you wish the objects width and height to be scaled to the screen

font ?
You should set the window to use InnerWidth and InnerHeight if you scale it so that images correctly fill the window.

List of Tags
Creation and deletion of the tags displayed in the listview on the left of the window is handled by the buttons below the listview.

ti_tag
This is the actual value of the tag that you wish to be given to the NewObject function. Set this to 1 for it to be igonored (TAG_IGNORE) or -1 and the Designer will put the text shown in the Tag listview in the code. Otherwise the number is included.

Select
This allows you to select the tag value and name from a list of known tags inside the Designer. The Designer does not select the correct tag type and does not guarantee the tag will do anything for your object. Some of these values are V39+ only so may have an effect on operating systems newer than yours if you are on V37.

Cloning Gadgets
It is, of course, possible to copy objects, just like gadgets, however any links to other objects are set to NULL in the copy. I did this for safety, as well as the fact that it is impractical to do otherwise.

Tag Type
Here you select what kind of tag you wish. At the moment there are the following types :

LONG
Pass the long value.

BOOLEAN
Pass the boolean value

STRING
Pass a pointer to the string, if the gadget strings are localized in this window then this string will be.

Array of BYTE
Pass a pointer to an array of bytes. These must be positive but you can enter negative values and they will be converted.

Array of WORD
Pass a pointer to an array of worda. These must be positive but you can enter negative values and they will be converted.

Array of LONG
Pass a pointer to an array of bytes. This has many uses, it is used in the BOOPSIDemo to create the ICA_Maps linking the string and proportional gadgets.

Array of STRPTR
This creates an array of string pointers, terminated with NULL, the MX and Cycle gadgets use this method to get their strings.

```
  These strings will be localized if the gadget strings are
 localized in this window.

List of STRINGS
  This creates a linked list of nodes, each of which points to a
 string in its ln_name field. These strings are localized as above.

User Structure
  This will insert text into the produced code referring to anything
 you like. This could be particularly useful if you wish to pass
 extra tags to the object when it is created by using the TAG_MORE
 tag. This is constant data, if you wish for non-constant then use
 User Type 2 as described below.

VisualInfo
  This will pass a VisualInfo to the object.

DrawInfo
  This will pass a DrawInfo to the object, some objects require this.

IntuiText
  An intuitext structure array will be set up, strings localized if
 required, the gadget font will be put in the intuitext font field.
  The coordinates are not scaled at the moment, perhaps this should
 be done.

Image
  This will pass a pointer to the image structure associated with
 the selected Designer image.

ImageData
  This will pass a pointer to the image data associated with a
 Designer image, this will be in chip ram.

Left Coord
  This passes the objects left coord, adding the border offset and
 scaling if required.

Top Coord
  This passes the objects top coord, adding the border offset and
 scaling if required.

Width Coord
  This passes the objects width, scaling if required, see above.

Height Coord
  This passes the objects width, scaling if required, see above.

Gadget ID
  This passes the GadgetID to the NewObject call.

Gadget Font
  This passes whichever font is to be used with the object, the
 screen font if that is what is selected in the code window.
  WARNING this is a TextAttr structure, not a TextFont structure
 as required by STRINGA_Font for example.
```

```
    Screen
      This causes the screen pointer to be passed.

    Object
      This allows a pointer to another Designer object to be passed,
     if this has not been created when it is needed then SetAttrs or
     SetGadgetAttrs will be used to set this at a later time.

    User Type 2
      This will insert text into the produced code referring to anything
     you like. This is a command or constant that the ti_data field is
     set to just before the objects creation.
      In assembly the code does a jsr to this piece of text, which has
     an XREF in the code. This function gets no parameters, and returns
     the ti_data filed in d0.

  Presets
    The menu on the object edit window allows you to select one of the
   preset object kinds. I have included the basic kinds and the V39
   colorwheel and gradientslider gadgets.
    To use these your program must open the gadgets/colorwheel.gadget
   and gadgets/gradientslider.gadget respectively.
    Please mail me suggestions for Presets, definitions would be
   appreciated.

   You really need to see the BOOPSI documentation before playing with
  these types.

   The preset border scrollers do not work properly if you have a
  different screen font, you will have to do something with User Type 2
  to fix this with a quick calculation.
```

1.28 Window code options

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

  Check if already open
    If this is set then the code checks to see if the window is already
   opened. This is necessary in most situations. Unless you are writing
   a program which needs multiple copies of the window, or can guarantee
   the window is opened only once then leave this set.
    If the window is checked and discovered open then several things can
   be made to occur.

    If Open MoveToFront
      Move the window in front of all other window.

    If Open Activate
      Activate this window.

    If Open Fail
```

Return an error from the open window function because the window has already been opened.

Only One Gadget Font
Having more than one gadget font can make windows look over-complicated and creates larger programs, but is sometimes required. You should normally only have one gadget font for each window, otherwise every gadget can have its own.
You must specify one font if you wish to use the screen font for your gadgets.

Producer Gadget Array
If you are going to need to read or change any gadget attributes then you need the gadget array to get a pointer to your gadget.

Open Only If Created Gads
Opening only if can create gadgets is a good idea.

Calculate Border Sizes
Different screen modes and fonts can result in different border sizes so you should set this to move the gadgets etc. to the same position relative to the borders as when they were designed.

Scale Using Screen Font
All gadgets have the screen font and scale all the Gadgets, Bevel Boxes, Images and Texts positions. It also scales the window size.
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 their are probably exceptions.
If you set this then you should select the window to use InnerWidth and InnerHeight so it its internal dimensions are scaled correctly.

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

Custom MsgPort
A custom message port can be supplied and the window will be closed safely, see
Multiple demo
for more information.

WorkBench AppWindow
WorkBench AppWindows allow icons to be dragged onto your window, if it is on the Workbench screen. It requires a seperate 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.

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 their are probably exceptions.

Params and Do Not Define Pointers
The params and " do not define some pointers " should be used in the same way as the MultipleDemo shows, these values can 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.

With V1.50 you now need to specify the parameters passed to the rend function associated with the window if one exists.

The rendwindow function requires the same extra parameters as the open function.

You must seperate the strings with "::", eg

```
" struct WindowNode * WinNode :: WinNode" in C
" pwinnode : pWindowNode :: pwinnode "    in Pascal
```

The second string will be inserted in the parameters of the rendwindow function whenever it is called.

See the multiple demo for an example.

Menus

This list allows you to select a Designer menu to attach to the window when it is opened. The following checkboxes allow you to modify the code created as follows.

Attach

Whether to attach the menu.

Create

Check if the menu has been created, if it has not then attempt to create it.

Fail

Fail to open the window if the menu is not created or fails to be created.

Free

Free the menu when the window is closed. This should not be done if the menu is used in more than one window.

Locale Options

Set these flags to specify which strings in this window you wish to be localized. Gadgets include data for listviews and initial text for text display gadgets.

See

locale

for more info.

SuperBitMap Window

This 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.

```
Create SuperBitMap
  This will create a bitmap for the window if the above option is set.
  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.

Slightly Comment Code
  This will mean that comments will be added to the functions relating
 to this window, but not others.
```

1.29 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 sizing 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
 If you are automatically creating a bitmap for a superbitmap window then
you should make the maximum sizes more reasonable because this is the size
of bitmap that will be created.
```

1.30 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.
         MENUPICK       : 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.
```

1.31 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.
 You can make some strange patterns with this, if you are bored, by
magnifying the magnify window.
```

1.32 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.
   DefPubName     : If you set PubScreenName then you can leave this
                    empty to take a parameter of the name, or specify
                    the name here.
   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 if DefPubName empty
                    otherwise it trys to open on the Name defined in
                    DefPubName.
   PubScrFallBack : Fall back to default screen if cannot find public
                    screen 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, they just wont do anything.

 For full information see manuals.
```

1.33 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.

1.34 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.
Images can now also be put in a window using BOOPSI Objects.

1.35 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 seperate 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 boxes you will get a normal box on a V37 machine.

With V1.50 of The Designer when you select the bevel boxes on the window when you are in bevel box creation mode. This changes the selected bevel box and allows you to use the buttons in the bevel box edit window on this bevel box. You cannot activate gadgets when this mode is in use.
You can also use the S, M and D keys to size, move and delete the selected bevel box. These are the only actions supported by bevel boxes at the moment. These buttons in the tools window carry out these actions as well, the others do nothing, at the moment.

1.36 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.

ALWAYS CHANGE THE LABELS

In V1.50 when you create new menus their are Xs all over the place in the labels. This is because their was to be a problem when you did not edit the labels for each title/item/subitem, which you should do. Code was producer like this :

```
#Define MyMenu_Title0 0
#Define MyMenu_Title0_Item1 1
```

Unfortunately, because of the first, the second became :

```
#Define 0_Item1 1
```

```
This only causes problems in C when you do not edit the labels and
                        #Define MyMenu2_Title0 0
                        #Define MyMenu2_Title0_Item1 1
becauses :
                        #Define 0_Item1 1
This case causes no conflict but it easily might have done, in C.

                    ALWAYS CHANGE THE LABELS
```

1.37 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 ehether 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 :

 Asm     :      jsr     MakeImages
                tst.l   d0
                bne     NoImages

                ..
                ..

                jsr     FreeImages
            NoImages:

 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
```

```
                */
                }

 Pascal  :   If MakeImages then
               Begin

                 { rest of program }

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

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, I think.
LoadRGB4 is used to set these to a viewport. To set a colourmap to a screen use :

```
   C          : LoadRGB4( &Scr->ViewPort, (UWORD *)colours, numcolours);
   Pascal     : LoadRGB4( @pscr^.viewport, pword(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.
Image palettes can now be used in screens as well.

The number of colosrs displayed in the image edit window it refers to the number of colours in the images palette, not the depth^2.

1.38 Locale Support

It is 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 versions of the catalog files.
 When you create new Windows and Menus the locale options are set depending on the preferences option "Localize Everything".
 In the localedemo you must remake the catalog files if you are not using Pascal because the producers create the strings in a slightly different order so Assembler and C versions would be disordered.

1.39 Screen Information

 The Designer allows you to create intuition screens for inclusion in your programs. Here is a description of all the options.

ScreenLabel
  Label of screen for use in source code

SA_Title
  Text at top of screen.

Localize Title
  Whether to set the screen title up as a localized string.

SA_Left
  Initial start x coord of screen, relative to overscan rectangle.

SA_Top
  Initial start y coord of screen, relative to overscan rectangle.

SA_Width
  Width of screen, can be larger or smaller than the overscan width,
  set to 65535 (STDSCREEENWIDTH) to be equal to overscan width.

SA_Height
  Height of screen, can be larger or smaller than the overscan height,
  set to 65535 (STDSCREEENHEIGHT) to be equal to overscan height.

SA_Depth
  Screen BitMap depth. Check to see the value you wish to use is valid
  for the screenmode and computers you wish your program to run on.
  Reducing this can result in loss of color information if you are
  using non-default values.
  Set this value to zero for the maximum screen depth for this screen
  mode.

SA_OverScan
  Overscan size, some may not fit on the screen.

Font
  Font type to use. SA_SysFont,0 is user preferred fixed width font,
  SA_SysFont,1 is user preferred, possibly proportional, font. Workbench
  uses SA_SysFont,1. SA_Font allows you to choose the font from the fonts
  directory to use.

Choose Font
  Choose font using font requester for SA_Font option above.

SA_Behind
  Open screen behind all other screens.

SA_Quiet
  Disable intuition rendering into screen. The screen will have no
  visible titlebar or gadgets, but depth and drag gadgets will still
  operate. To completely disable rendering into screen all windows
  should have WFLG_RMBTrap tag set.

SA_ShowTitle
  This means that the screens title bar will be in front of the title
  bars of any backdrop window.

SA_AutoScroll
  Screen moves as mouse reaches edge of display clip.

SA_BitMap
  Use a custom bitmap for the screen. This is disabled if you set
  the V39 SA_LikeWorkBench tag to true.

CreateBitMap
  Make code to create bitmap when screen opened. UserData contains bitmap
  pointer for freeing. You must free this when you close the screen. If
  you use this do not specify width and height as 65535.

SA_Colors
  This list of all the images loaded into the Designer allows you to get
  a colour map from the image and use it for your screen. The map is
  copied so you can delete the image without losing the palette.
  You can also get the CMAP from an IFF ILBM using the menu option,
  Get CMAP.

Default
  Clear custom palette from screen.

ID
  The display ID of the screen is set here, see the include files for the

proper values. In V39 and above you can use the ScreenReq button to choose from the available modes, this list is not exhaustive though so you can input an 8 digit hexadecimal number. Here are a few of the standard ones

```
00000000        LORES_KEY
00008000        HIRES_KEY
00008020        SUPER_KEY
00000800        HAM_KEY
00000004        LORESLACE_KEY
00008004        HIRESLACE_KEY
00008024        SUPERLACE_KEY
00000804        HAMLACE_KEY
```

These screen modes are machine independant, if you use these you will get PAL or NTSC depending on the machine you are using.

If you use the screen requester you may get a monitor ID built in to the Display ID, so if you have a PAL machine it may not open on an NTSC machine and vice versa.

Type
Would you like a custom or public screen.

PubName
Name you wish to give to public screen, this implies the screen is a public screen. The name must be unique.

Do SA_PubSig
You must pass a parameter to the open screen function which is the signal you wish to have set when all the windows on the screen are closed. Only set if pubname is not empty or Screen Type is Public.

SA_Pens
Choose the value of the pens for your screen, you can only edit those that your system uses, so V37 cannot edit the 3 which only exist in V39+. Try swapping ShinePen and ShadowPen for confusion.

Default Pens
Use default pens, ignore changes.

SA_FullPalette
Use full WorkBench palette for screen rather than V34 subset.

Also available are more tags in the menu attached to the edit window, these are mainly V39+ only tags.

SA_ErrorCode
If the window fails to open then the long (screen label)Error contains the error code returned by openscreen.

SA_SharedPens (V39)
Screens that wish to manage their own pens should set this tag.

SA_Draggable (V39)
Specify whether the screen is draggable, you should have a good reason to disable this.

```
SA_Exclusive (V39)
  Specify screen will not share display.

SA_Interleaved (V39)
  Request interleaved bitmap for screen.

SA_LikeWorkBench (V39)
  Screen clones workbench screen dimensions, depth, colours, ID etc.
 If the intuition.library is less than version 39 then this tag is
 not set and the defined width, depth etc. are used. For this reason
 you should not set a specific monitor ID but use general display IDs
 such as 00008000 for Hires. See the list above. You should consider
 the defined data as a default screen specification, if unable to open
 a screen the same as the WorkBench screen.

 This disables
   SA_Left          When Version > 38
   SA_Top           When Version > 38
   SA_Width         When Version > 38
   SA_Height        When Version > 38
   SA_Depth         When Version > 38
   SA_DisplayID     When Version > 38

   SA_BitMap        Always disabled if SA_LikeWorkBench
   CreateBitMap     Always disabled if SA_LikeWorkBench

  I think this is the correct handling, it is what is implemented
 for now.

 The screen edit window can be made to open on the screen, if possible,
by setting the preference option "Screen editor on edit screen",
otherwise the edit window appears on the default public screen.
```

1.40 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 V1.2  ( Available on Aminet )

   C= ( The development teams )
```

for the Amiga.

1.41 Producers

AsmProducer

CProducer

PasProducer
All code is generated from saved designer files by the Producers , their
are now three of these, Assembler C and Pascal. The generate button saves
the current data (which must have already been saved) and then runs the
selected producer on the saved file.
The producers can also be run from workbench or CLI with their respective
methods of passing parameters, multiple files are supported.
Producers should be in the same directory as The Designer.

Fonts

Code Options

Preferences

1.42 AsmProducer

Compilation :
I have tested the source with Devpac2 and A68K, linking with BLINK, for
example to make the AllKindsDemo do the following

```
AsmProducer AllKindsDemo.des
A68K AllKindsDemoMain.s
A68K AllKindsDemo.s
BLINK AllKindsDemoMain.o AllKindsDemo.o TO AllKindsDemo
AllKindsDemo
```

or with Devpac2

```
AsmProducer AllKindsDemo.des
genim2 -l AllKindsDemoMain.s
genim2 -l AllKindsDemo.s
BLINK AllKindsDemoMain.o AllKindsDemo.o TO AllKindsDemo
AllKindsDemo
```

I would suggest that comments are always made to inform you of the
parameters that each function takes, they also make the code more clear.

Main
If you select to Produce a main program then code will be created that
will be able to run, open the first window on the window list, and first

screen if you wish. It will have a minimal Message handler to cope with window closing. Use of window and menu IDCMP handlers can enable you to use this as a very simple, ready-made, program. See
Demos

Libraries
See Library help for information on this.

Windows
At least two functions will be created for each window
Open(window label)Window
Close(window label)Window

OpenWindow allocates everything, opens, attaches menus and all sorts of other things. It can fail for any number of reasons.
Return codes : 0 Success
Non-0 Failure (I cannot remember them all)
(look at your code for yours)

Parameters, if required are as follows
a0 MsgPort if uses custom MsgPort
a1 Screen if uses PubScreen or PubScreenName or CustomScreen.
a2 AppWin MsgPort if AppWindow
a3 AppWin ID if AppWindow
a4 BitMap if custom BitMap required and not created.

CloseWindow frees all Designer code allocated memory and closes down everything cleanly.

Also possible is Rend(window label)Window which draws BevelBoxes, places text and draw images on the screen as graphical data. This needs to be refreshed when the window is sized, depth arranged etc.

Menus
The function MakeMenu(Menu-Label) takes a parameter of the ↩ Screen
VisualInfo in a0 and returns an error code. The menu pointer, if successfull, is in (Menu Label). Multiple copies can be made, but you must remember to free all.
Return codes : 0 Success
1 CreateError
2 LayoutError

Images
If there are any images imported into the Designer then two ↩ functions
will be created.

Screens
Each Designer Screen causes one function to be created. This is ↩ called
Open(Screen Label)Screen and may take parameters. If you specify DoPubSig then the function will require the Signal allocated for the

```
screen with AllocSignal in d0.
 If you wish for a custom bitmap and do not wish the code to create its
own then a0 must contain a pointer to a BitMap structure properly
initialized. If you wish the code tocreate this BitMap for you then you
must free it, a pointer to this bitmap will be in the screen UserData
field.
 The Screen is returned in d0, which is zero on failure.
```

1.43 CProducer

```
                All the produced code has been tested with DICE C 2.07.56R.

 I have used GCC 2.5.8 to compile the source code and have removed the
few errors found, but unfortunately the archive I obtained had no Amiga
libs so I cannot link the programs yet, so this is not fully tested.

 If you find any problems with the produced code with any compiler please
send me details of the exact problem and the solution so that I can try
and fix it.

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.
 The Designer will cannot totally compatible with GTB, but this should
help.

 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.

Main
 If you select to Produce a main program then code will be created that
will be able to run, open the first window on the window list, and first
screen if you wish. It will have a minimal Message handler to cope with
window closing. Use of window and menu IDCMP handlers can enable you to
use this as a very simple, ready-made, program. See
              Demos

              Windows
                 At least two functions will be created for each window
Open( window label )Window
Close( window label )Window

 OpenWindow allocates everything, opens, attaches menus and all sorts of
other things. It can fail for any number of reasons.
   Return codes :      0     Success
                       Non-0 Failure     ( I cannot remember them all  )
                                         ( look at your code for yours )

Parameters, if required are as follows
```

```
        MsgPort if uses custom MsgPort
        Screen if uses PubScreen or PubScreenName or CustomScreen.
        AppWin MsgPort if AppWindow
        AppWin ID if AppWindow
        BitMap if custom BitMap required and not created.

 CloseWindow frees all Designer code allocated memory and closes down
everything cleanly.

 Also possible is Rend( window label )Window which draws BevelBoxes,
places text and draw images on the screen as graphical data. This needs
to be refreshed when the window is sized, depth arranged etc.


              Menus
                 The function MakeMenu(Menu-Label) takes a parameter of the  ↩
                     Screen
VisualInfo and returns an error code. The menu pointer, if
successfull, is in (Menu Label). Multiple copies can be made, but you
must remember to free all, the creation routine does not check if the
menu has already been created.
   Return codes :      0  Success
                       1  CreateError
                       2  LayoutError


              Images
                 If there are any images imported into the Designer then two  ↩
                     functions
will be created if __CHIP is not specified in the code window, if it
is then these functions are not required.


              Screens
                 Each Designer Screen causes one function to be created. This is ↩
                      called
Open( Screen Label )Screen and may take parameters. If you specify
DoPubSig then the function will require the Signal allocated for the
screen with AllocSignal.
 If you wish for a custom bitmap and do not wish the code to create its
own then you must supply a pointer to a BitMap structure properly
initialized. If you wish the code tocreate this BitMap for you then you
must free it, a pointer to this bitmap will be in the screen UserData
field.
 The Screen is returned in by the function, NULL for failure.
```

1.44 PasProducer

```
               New:
 HSPascal V1.2 is now available with V3.1 units, these need slightly
different code so you must set the code option to allow for this. None
of the demos have this set so to compile the demos you must change this.
 The classic error these units produce without this flag set is that
```

```
 ActivateWindow no longer returns a value so if your error line has this
 command in then you should check this flag.
  Contact HiSoft for details on upgrading.

Compilation:
  Run the pasproducer on the .des file then load the main file into the
 editor and press Ctrl-B to build all units, then run.


              Windows
               Functions:
   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);
     procedure freebitmap(pbm:pbitmap;width,height:word);
                          ^ bitmap structure must be allocated with
                            allocmem
```

```
              Menus
                 The function MakeMenu(Menu-Label) takes a parameter of the  ↩
                     Screen
 VisualInfo and returns boolean. The menu pointer, if
 successfull, is in (Menu Label). Multiple copies can be made, but you
 must remember to free all, the creation routine does not check if the
 menu has already been created.


              Images
                 If there are any images imported into the Designer then two  ↩
                     functions
 will be created.


              Screens
                 Each Designer Screen causes one function to be created. This is ↩
                      called
 Open( Screen Label )Screen and may take parameters. If you specify
 DoPubSig then the function will require the Signal allocated for the
 screen with AllocSignal.
  If you wish for a custom bitmap and do not wish the code to create its
 own then you must supply a pointer to a BitMap structure properly
 initialized. If you wish the code to create this BitMap for you then you
 must free it, a pointer to this bitmap will be in the screen UserData
 field, freebitmap should be used for this.
  The Screen is returned by the function, nil for failure.
```

1.45 Demonstration Programs

```
                The following demos are not edited at all from the Designer  ↩
                   Produced
code :

  LocaleDemo
  AllKindsDemo
  ScreenDemo
  PubScreenDemoP2
  BOOPSIDemo

 The rest are all based on the main programs and have been modified
to demonstrate the capabilities of the Designers code. The main
file is only the shell of a program, to make it do anything you must
fill in the gaps.

LocaleDemo
  The locale demo catalog files need to be re-made to work with the Asm
 and C version of the code, this is because the supplied ones work with
 the executable supplied (Pascal) and the strings are in a different
 order.
  The default language of this demo is German, if you see any other then
 locale is working. French and English catalogs are made so far.

MultipleDemo
```

Demonstrates ability to open multiple copies of windows using several features of the Designer. Each language needs a different .des file for this demo.

ToggleDemo
Uses GT_SetGadgetAttrs to change state of text gadget, and modify state of Boolean gadgets so you have a set of MX Boolean gadgets.

ShowIconDemo
Drag Icons into the window to display the path of the directory lock associated with the icon and the filename, if it exists.

KeyDemo
Shows use of keys for gadgets. Correct operation for each of the gadget kinds is used, as officially defined.

AllKindsDemo
All the gadget kinds are created on this window, along with texts, images and bevel boxes. The window scales to the screen font.

ScreenDemo
Opens up a Hires screen with a custom palette and pens displaying an image in a window.

PubScreenDemoP1
PubScreenDemoP2
A demo in two parts. P1 opens up a Hires screen with a custom palette as a public screen, with a window on it containing a description of the demo.
P2 will try to open a window on the public screen, if it is not set to private. P1 must be running for P2 to succeed. Setting private will not be effective if there is a visitor window on the screen.

SuperBitMapDemo
This demo opens a SuperBitMap window, allocating the bitmap itself, and allows you to draw on it. This window does not actually have to be SuperBitmap for this demo, but it may in the future. This demo will open on the public screen created by PubScreenDemoP1, if it exists, otherwise on the default Public Screen.

BOOPSIDemo
This demo opens up a window with several objects on it, including images making a requester type look and a proportional gadget and a string gadget linked together. I was asked about this kind of background imagery so here it is.

AsmProducer

CProducer

PasProducer