

# GINO

## *getting started*

*version 6.5*

BRADLY ASSOCIATES LTD  
Manhattan House  
140 High Street  
Crowthorne  
Berkshire RG45 7AY  
England  
Tel: +44 (1344) 779381  
Fax: +44 (1344) 773168  
support@gino-graphics.com  
www.gino-graphics.com



Information in this manual is subject to change without notice.

While Bradly Associates Ltd. makes every endeavour to ensure the accuracy of this document, it does not accept liability for any errors or omissions, or for any consequences arising from the use of the program or documentation.

GINO getting started version 6.5

© Copyright Bradly Associates Ltd. 2005

All rights reserved.

All trademarks where used are acknowledged.

# Contents

---

Product Description	5
Installation	6
PC Installation	6
UNIX, Linux and OpenVMS Installation	6
F90 Version	7
Building the F90 Example Programs in a GUI environment	7
Building the F90 Example Programs in a Command Prompt Window	7
Porting from F77-GINO	8
Example Fortran Program	12
SCALE routine	12
C++ Version	12
Building the C++ Example Programs in a GUI environment	12
Building the C++ Example programs in a Command Prompt Window	12
C++ Language Features	13
Example C Program	15
Delphi Version	15
Delphi Language Features	16
Visual Basic Version	16
VB Language Features	16
Visual Studio .NET Version	16
VS.NET Language Features	16
Differences between PC, UNIX and OpenVMS versions	17
Your First Program	18
Printing	19
The Structure of a simple non-GUI program	19
The Structure of a simple GINOMENU GUI program under Windows	20
The Structure of a simple non-GINOMENU GUI program under Windows	20
GINO Coordinate System	21
Origin of Coordinate System	22

GINO Configuration File	22
Windows	23
UNIX and Linux	23
OpenVMS	23
Internal Naming Convention	23
Error Handling	24
Redistributing GINO files	25
PC Redistribution	25
UNIX, Linux and OpenVMS Redistribution	26

# Chapter

# 1

---

---

## GETTING STARTED

This manual gives a brief overview of using the different components of GINO. Language considerations, porting hints, the structure of a simple program and re-distributing programs are all covered.

---

## Product Description

GINO is a suite of programmable toolkits for creating professional graphics and GUI applications. It is available for a wide variety of languages on all major platforms:

### **GINO**

GINO is the graphics engine and provides all the low-level features such as basic drawing, characters & fonts, colours and filling, advanced features such as interaction, hierarchical segment structures, widowing & masking and full 3D features such as 3D wire-frame drawing, bezier surfaces, lighting & shading and texture mapping.

GINO's 3D functionality features approximately 75% of the OpenGL standard with portable routine names that can be used across all GINO platforms.

### **GINOGRAF**

GINOGRAF provides 2D and 2.5D graphs and charts including X-Y plots, histograms, barcharts, stepcharts, vector-diagrams, polar charts and pie-charts.

### **GINOSURF**

GINOSURF provides 3D contour maps, surfaces and cross-sections and can utilise GINO's OpenGL driver for animating and lighting surfaces in real-time.

### **GINOMENU**

GINOMENU provides a programmable graphical user interface including features such as text/value arrays, tree-views, combo-boxes, graphics frames, rich-text editor, TTY emulator, docking panes and other commonly used widgets.

### **GINOMENU Studio**

GINOMENU Studio is a RAD interface to GINOMENU enabling GUI applications to be created with the minimum of programming effort.

All products are compatible with each other and can be used at the same time. GINOGRAF and GINOSURF use the underlying features of GINO, whereas GINOMENU can be used as a standalone product, so in this guide we will show the differences between creating a GINO application and a GINOMENU application.

---

## **Installation**

### **PC Installation**

The GINO CD for the PC platform is a multi-product/multi-compiler CD. All products for all compilers are supplied on the one CD and the Serial Number is used to install the appropriately licenced software. The Serial Number can also contain a time-limiting date string and this gives the possibility of evaluating additional software simply by obtaining a Serial Number for the appropriate packages required.

Because the CD's are all mastered at the beginning of the release, patch files will be released regularly on the GINO web site.

The GINO CD uses a standard installation procedure which is self-explanatory. At the end of the installation procedure you are asked whether to automatically update AUTOEXEC.BAT (Win9x) or the System Registry (WinNT/2K/XP) so that environment variables can be set up for using GINO. If you choose not to, Win9x users will need to refer to the file GINOVARS.BAT and WinNT/2K/XP users will need to refer to the file NT2000XPVARS.TXT for help in setting up the environment variables before GINO can be run.

### **UNIX, Linux and OpenVMS Installation**

The UNIX/Linux/OpenVMS CD is a multi-product/multi-platform CD. Instructions for installing the software are printed on the inside of the CD inlay card and the Serial Number contains the information needed to load the appropriately licenced software.

---

## F90 Version

### Building the F90 Example Programs in a GUI environment

GINOMENU Studio examples are found in the folder `\examples\mstudio` and when running GINOMENU Studio, the initial default working directory is set to this location. If using Developer Studio, Visual Studio or any other Fortran development environment, refer to the notes in the 'running instructions' file and use the GINO example programs found in the `\gino`, `\graf`, `\surf`, `\menu`, `\mstudio` and `\demo` folders stemming from the `\examples` folder.

### Building the F90 Example Programs in a Command Prompt Window

GINO's example programs are found in the `\gino`, `\graf`, `\surf`, `\menu`, `\mstudio` and `\demo` folders (depending on the products licenced), stemming from the `<gino installation>\examples` folder (where `<gino installation>` is by default: `\Program Files\gino\v6.5`). To build the example programs in a DOS box or Command Window, the following batch files are provided:

GINOWBLD	For GINO-F, GINOGRAF & GINOSURF WinAPI programs
GINOBLD	For GINO-F, GINOGRAF & GINOSURF OpenGL programs
MENUBLD	For GINOMENU programs (with/without the other packages)

(The above batch files are also supplied for double-precision versions of GINO and have a 'D' appended to the end, e.g. GINOWBLDD).

The batch files reside in the top-level GINO directory but will be picked up automatically provided the system PATH has been updated as per the installation instructions.

To compile, link and run an example program, move into one of the example directories and run the batch file as follows:

```
C:> cd \Program Files\gino\v6.5\examples\graf
C:> ginowbld grafex1
```

or

```
C:> cd \Program Files\gino\v6.5\examples\gino
C:> ginoobld ginoex10
```

or

```
C:> cd \Program Files\gino\v6.5\examples\menu
C:> menubld hello
```

OpenGL example programs supplied with GINO are as follows:

```
ginoex2.f90, ginoex10.f90, ginoex11.f90 (GINO)
surfex5.f90, surfex6.f90 (GINOSURF)
draw.f90, performance.f90, bottle.f90, spingl.for, room3d.f90 (GINOMENU)
fan.f90, designer.f90, solarsystem.f90, familytimeline.f90, cutandfill.f90,
funcview.f90, surfview.f90, trisurf.f90 (GINOMENU STUDIO)
```

## Porting from F77-GINO

Prior to version 5.0, GINO was supplied as either an F77 binding or F90 binding with appropriate documentation. The F77 binding has now been frozen and all documentation refers to the F90/C++ binding. However, the F90 software libraries still include the F77 routines, so software written using the F77 routines will continue to work as before and there are no plans to discontinue the support of these routines.

All F90/95 compilers support the F77 language, so users with existing F77 programs can port their program to an F90 compiler and gradually introduce new GINO calls without there being any incompatibility problems. For existing F77 users who are unclear how to use the new F90 language features of GINO, they are explained below:

Cross-References from the F77 name to the F90 name are provided in each product manual, but automatic conversion software is not possible due to routines not always having a one-one mapping and some of the routines using a different number of arguments.

If you call the F77 routines with an F90 compiler you can also make use of supplied module files to perform parameter checking. The F77 files are named: xxxx\_F77.MOD and the file is used by inserting a 'use' statement at the beginning of your Fortran source: Note that when writing fixed-format code, the use statement must begin in column 7. (Refer to the F90 section entitled 'Long descriptive Names' for more details on how to use Modules in your program).

While F90/95 is in fact case insensitive, the GINO manuals make use of mixed case to ease readability.

## Long descriptive Names

The long names can be up to 31 characters long and each package prefixes its routine names with the following:

GINO	names prefixed with lower-case g
GINOGRAF	names prefixed with lower-case gg
GINOSURF	names prefixed with lower-case gs
GINOMENU	names prefixed with lower-case gm

The long-names are supplied in an F90 Module file and to use this file, a USE statement is required in every MAIN and SUBROUTINE that contains GINO calls. Each package requires a separate USE statement as follows:

GINO	use gino_f90
GRAF	use graf_f90
SURF	use surf_f90
MENU	use menu_f90

This USE statement should be placed immediately after the PROGRAM or SUBROUTINE statement. The module file is picked up in a compiler-dependent way, but generally requires a switch when compiling, e.g. /module or -mod followed by the pathname. Refer to your compiler documentation or the GINO supplied batch files for more details.

Not all GINO routines have a one-one mapping from F77 to F90. Some routines have a straightforward mapping of name and argument list, for example:

```
CALL CHASTR (STRING)
call gDisplayStr (string)
```

Others, have a one-one mapping of the name, but the argument list varies:

```
CALL CURSOR (KEY, X, Y)
call gGetCursorEvent (key, point)
```

where X,Y has been replaced by **point** which is a structure of a derived type called GPOINT (see later section).

Other F90 routines are equivalent to two or more F77 routines, for example:

```
CALL ARRATR (AXISW, AXISH, STYLE, XJUST, YJUST, XTEXT, YTEXT)
CALL TEXARR (PARENT, XPOS, YPOS, COLS, ROWS, IDENT)
CALL WIDHLP (WIDGET, FLAG, HELP)
```

is equivalent to:

```
call gmCreateTextArray (parent, xpos, ypos, width, height, cols, rows...)
```

where ... refers to Optional Arguments that can be used to set other attributes.

### Pre-defined Constants

Pre-defined constants are included in GINO to make programs more legible. All pre-defined constants are in upper-case and begin with the letter G. They are all defined in the appropriate Module file, so the USE statement is required as mentioned above.

A typical example would be:

```
CALL LINCOL (2)
```

is replaced by

```
gSetLineColour (GRED)
```

### Derived Types and Structures

A Derived Type is a combination of Intrinsic Types. A scalar object of this new Type is then called a Structure. For example the F90 equivalent of the routine CURSOR is gGetCursorEvent which uses a structure called **point** which is a derived type called GPOINT. The type declaration statements for all GINO's derived types are contained in the Module files (and listed in an Appendix of each manual), in this case:

```
type GPOINT
  real :: x
  real :: y
end type
```

and this does **not** need to be coded by the user, but to create the structure, a type statement is required:

```
type (GPOINT) point
```

this statement needs adding with other declaration statements at the top of the program. As seen from the type declaration statement, **point** will now consist of an x element and y element and these are accessed using the character '%'

```
xpos = point%x  
ypos = point%y
```

Note that elements of structures in the C language are accessed using the character '.' e.g.

```
xpos = point.x;  
ypos = point.y;
```

The '.' character is used in many examples in the GINO manuals and some Fortran compilers do accept this character. However it is not strict Fortran, therefore we recommend that the '%' character should always be used when programming in Fortran.

When passing data into a structure, this can be done in one statement as follows:

```
point = GPOINT(10.0,20.0)
```

### Optional Arguments

Optional Arguments are arguments that may or may not be defined when calling a subroutine. All GINO optional arguments begin with the following letter:

GINO	names prefixed with lower-case g
GINOGRAF	names prefixed with lower-case gg
GINOSURF	names prefixed with lower-case gs
GINOMENU	names prefixed with lower-case gm

A typical routine would be:

```
gmCreateStatusBar(parent, gmPanels, gmSizes, gmLabel, gmDragDrop)
```

which has one argument which must be given a value and a further 4 arguments which are optional. When calling a routine with optional arguments, the optional arguments must be declared using a keyword which is the same as the argument name. e.g

```
status = gmCreateStatusBar(IPAREN, gmLabel='Ready')
```

As can be seen, gmPanels and gmSizes have been omitted and the argument gmLabel is used with the format 'keyword=value', so that the subprogram knows which arguments are being passed.

One of the strengths of using Optional Arguments is that when more functionality is required in a GINO routine, more arguments can be added at a later release.

## Example Fortran Program

An example of a program using all of the new features is as follows:

```

program newprog
use gino_f90
type (GPOINT) point
call gOpenGino
call gMwin
call gGetCursorEvent(key,point)
call gSetLineColour(GBLUE)
call gDrawLineTo2D(point%x,point%y)
call gCloseGino
stop
end

```

## SCALE routine

The SCALE command is a Fortran 90 intrinsic, so the use of the GINO-F command SCALE2 is recommended instead of SCALE to avoid any name clash.

---

## C++ Version

### Building the C++ Example Programs in a GUI environment

GINOMENU Studio examples are found in the folder \examples\mstudio and when running GINOMENU Studio, the initial default working directory is set to this location. If using Developer Studio, Visual Studio or any other C++ development environment, refer to the notes in the 'running instructions' file and use the GINO example programs found in the \gino, \graf, \surf, \menu and \mstudio folders stemming from the \examples folder.

### Building the C++ Example programs in a Command Prompt Window

GINO's example programs are found in the \gino, \graf, \surf, \menu and \mstudio folders (depending on the products licenced), stemming from the <gino installation>\examples folder (where <gino installation is by default: \Program Files\gino\v6.5). To build the example programs in a DOS box or Command Window, the following batch files are provided:

GINOWxBLD      For GINO-F, GINOGRAF & GINOSURF WinAPI programs

GINOxBLD	For GINO-F, GINOGRAF & GINOSURF OpenGL programs
MENUxBLD	For GINOMENU programs (with/without the other packages)

where x=M (Microsoft), B (Borland) or A (Absoft).

The batch files reside in the top-level GINO directory but will be picked up automatically provided the system PATH has been updated as per the installation instructions.

To compile, link and run an example program, move into one of the example directories and run the batch file as follows:

```
C:> cd \Program Files\gino\v6.5\examples\graf
C:> ginowmbld grafex1      (for building under Visual C/C++)
```

or

```
C:> cd \Program Files\gino\v6.5\examples\gino
C:> ginoobld ginoex10      (for building under Borland C)
```

or

```
C:> cd \Program Files\gino\v6.5\examples\menu
C:> menumbld hello        (for building under Visual C/C++)
```

OpenGL example programs supplied with GINO are as follows:

```
ginoex2.cpp, ginoex10.cpp, ginoex11.cpp (GINO)
surfex5.cpp, surfex6.cpp (GINOSURF)
```

## C++ Language Features

The C/C++ version of GINO uses features such as Predefined constants, Structures and Optional Arguments. C/C++ is case sensitive so routine names spelt using the wrong case will show up as an 'Unresolved External Symbol'.

The C interface is declared in header files which must be referenced as follows:

GINO	#include <gino-c.h>
GINOGRAF	#include <graf-c.h>
GINOSURF	#include <surf-c.h>
GINOMENU	#include <menu-c.h>

The include file is found using the `/I` switch and by default the GINO include files can be referenced using: `/I%GINO%\include` where `%GINO%` points to the top-level GINO directory. Refer to your compiler documentation or the GINO supplied batch files for more details.

## Structures

The structure definition statements for all GINO's structures are contained in the Include files (and listed in an Appendix of each manual), for example:

```
typedef struct {  
    float x;  
    float y;  
} GPOINT;
```

This does **not** need to be coded by the user, but to create the structure, the following statement is required:

```
GPOINT point;
```

As seen from the structure definition statement, **point** will now consist of an `x` element and `y` element and these are accessed using the `!` character:

```
xpos = point.x;  
ypos = point.y;
```

When passing data into a structure, this can be done in one statement as follows:

```
GPOINT point = {10.0, 20.0};
```

## Variable Argument Lists

Variable Argument Lists or Optional Arguments are arguments that may or may not be defined when calling a subroutine. All GINO optional arguments begin with the following letter:

GINO	names prefixed with lower-case g
GINOGRAF	names prefixed with lower-case gg
GINOSURF	names prefixed with lower-case gs
GINOMENU	names prefixed with lower-case gm

A typical routine would be:

```
gmCreateStatusBar(parent, gmPanels, gmSizes, gmLabel, gmDragDrop);
```

which has one argument which must be given a value and a further 4 arguments which are optional. When calling a routine with optional arguments, the optional arguments must be declared using a keyword which is the same as the argument name. e.g

```
status = gmCreateStatusBar(IPAREN, gmLabel,"Ready",0)
```

As can be seen, gmPanels and gmSizes have been omitted and the argument gmLabel is used with the format 'keyword,value', so that the subprogram knows which arguments are being passed. Also note that a routine with a variable argument list should be terminated with a single NULL even if none of the optional arguments are actually being used.

One of the strengths of using Optional Arguments is that when more functionality is required in a GINO routine, more arguments can be added at a later release.

## Example C Program

An example of a program using Structures and optional arguments is as follows:

```
#include <gino-c.h>
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpszCmdParam, int nCmdShow)
{
GPOINT point;
int key=0;
    gOpenGino();
    gMwin(hInstance, hPrevInstance);
    gGetCursorEvent(&key, &point);
    gSetLineColour(GBLUE);
    gDrawLineTo2D(point.x,point.y);
    gSuspendDevice();
    gCloseGino();
    return 0;
}
```

---

## Delphi Version

Delphi example programs are found in the folders \gino, \graf and \surf stemming from the \examples folder.

---

## Delphi Language Features

All of GINO's pre-defined constants and structures (records) have been declared in the GINO interface files and can be used as per the main documentation however due to the restriction in Delphi where optional arguments (varargs) can only be used with External procedures using the cdecl calling convention, all GINO optional arguments must be specified at this release.

For further Delphi specific documentation, refer to the chapter in the GINO manual entitled "GUI Programming" and also refer to the Delphi Running Instructions file.

---

## Visual Basic Version

VB6 example programs are found in the folders \gino, \graf and \surf stemming from the \examples folder.

### VB Language Features

All of GINO's pre-defined constants and structures (records) have been declared in the GINO interface files and can be used as per the main documentation however due to the restriction in VB where optional arguments cannot be user-defined types, all GINO optional arguments must be specified at this release.

For further VB6 specific documentation, refer to the chapter in the GINO manual entitled "GUI Programming" and also refer to the VB Running Instructions file.

---

## Visual Studio .NET Version

Visual Studio .NET example programs are found in the folders \vbasic, \csharp, \cplusplus and \jsharp stemming from the \examples folder.

### VS.NET Language Features

Whether programming in VB.NET, C#.NET, J#.NET or C++.NET, the interface to GINO is exactly the same and is via the same set of Assemblies (DLLs), so no further libraries are required if changing from one language to another at a later stage.

The names of all GINO routines have had to change in the .NET interface due to the introduction of class assemblies and now, instead of being prefixed by g, gg and gs, routine names are prefixed by gino., graf. and surf.  
e.g.

gOpenGino becomes gino.OpenGino

GINO's pre-defined constants are handled as Enumerators and structures are used as per the main documentation. Due to the restriction of optional arguments, these must be specified at this release, however multiple interfaces to the same routine have been provided for some routines via a drop-down in the GUI interface, so that only a restricted number of arguments need to be declared.

For further .NET specific documentation, refer to the chapter in the GINO manual entitled "GUI Programming" and also refer to the relevant .NET Running Instructions file.

---

## Differences between PC, UNIX and OpenVMS versions

Porting a GINO program from a Workstation platform to the PC or vice-versa is relatively straightforward providing the program has been written with portability in mind.

The only GINO library that includes platform-specific routines is GINOMENU with GINOMENU for Windows providing many features that are not available in GINOMENU-X. The differences between the two packages are listed at the beginning of the GINOMENU-X manual.

All other GINO libraries are 99% routine-name portable (with the exception of the .NET interface as explained earlier), with the main exception being the Device Nomination routine (gMwin, gXwin, etc.) as detailed in the section Your First Program.

When writing a GINO program that is to be portable, bear in mind that some devices have more hardware features than others, so make full use of the enquiry routines to establish the best use of available resources on each device. Areas to watch are:

- Available Drawing Area
- Number of Colours
- Linestyles and Hatchstyles
- Fonts
- Image output capability
- Cursor Types/Actions

---

## Your First Program

GINO can be used to create GUI programs, (either using GINOMENU plus other GINO calls if programming in Fortran or C++, or by using GINO calls within a GUI RAD environment such as Delphi, VB or .NET) or legacy-style non-GUI programs.

The programming style of a non-GUI program is generally linear and a GUI program revolves around an event or action loop and therefore its structure looks quite different. If using GINOMENU, this is explained in more detail in the GINOMENU manual and if using a RAD tool, the event mechanism is already built-in, but here we will look at how to begin each type of program. After the call to `gOpenGino`, the next call in a GINO program is the call to a Device Nomination routine and this differs depending on what type of program is being created:

<code>call gMwin</code>	non-GINOMENU program under Microsoft Windows
<code>call gWogl</code>	non-GINOMENU OpenGL program under Microsoft Windows
<code>call gXwin</code>	program under X Windows
<code>call gGlx</code>	OpenGL program under X Windows
<code>call gGuiwin</code>	GINOMENU program under Microsoft Windows
<code>call gOglwin</code>	GINOMENU OpenGL program under Microsoft Windows

Note that when creating GUI programs under X Windows, because the complete GUI is contained within one X Window using GINOMENU's GUI emulation, the nomination routines of `gXwin` and `gGlx` still apply.

Device Nomination routines are grouped in families and different members of each family may be required depending on the programming environment. For example, Delphi and .NET use `gMwindc` and `gWogldc`. Further details can be found in Appendix B or the GUI Programming Chapter of the GINO User's Guide.

## Printing

When GINO output is to be sent to a printer or metafile, a non-GUI program (or X Windows program) needs its Device Nomination routine replacing with a printer routine such as gEps or gPng. In the case of a GUI program, because only the graphics part of the program is sent to the printer, and the GUI elements are left on the screen, a generic printer routine such as GINOMENU's gmPrinterControl() is used in the middle of the program to direct the graphics away from a graphics window and to the Windows Printer Dialog. Refer to Appendix B of the GINO User's Guide for details of GINO printer drivers, the Graphics Frames chapter of the GINOMENU User's Guide for printing within GINOMENU or the GUI Programming Chapter of the GINO User's Guide for printing within other RAD tools.

## The Structure of a simple non-GUI program

The essential GINO routines required by any program are concerned with control of the hardware device used to display graphical output. Before being used to control hardware devices, GINO must be initialized. Each output device used must be initialized before use, and released after use. If more than one output device is to be used by a program, each device must be released or suspended before the next device is initialized.

The sequence of operations in driving an output device is as follows:

gOpenGino ()	This must be the first graphics call in any GINO program. It initializes GINO, and is only required once in a program.
xxxxxx ()	Where XXXXX is replaced by the name of the output device for the following graphics generation. This call initializes the output device ready for output from the user's program.
Graphics Calls	Calls to graphics routines from GINO, GINOGRAPH or GINOSURF to produce the required output.
gCloseDevice ()	This call releases the output device just used. Failure to use this call may leave the device in an undesirable state.
gCloseGino ()	This call closes down GINO. It includes an implicit call to gCloseDevice() to ensure a tidy shutdown.

The sequence between gOpenGino and gCloseGino can be repeated to initialize and close down GINO for as many devices as are required.

## The Structure of a simple GINOMENU GUI program under Windows

A GINOMENU program under Windows still has to initialize GINO and an output device, but then initializes GINOMENU and sets itself up for continually looping around an ACTION loop until program termination whereby it closes down GINOMENU and GINO in the usual manner:

<code>gOpenGino()</code>	This must be the first graphics call in any GINO program. It initializes GINO, and is only required once in a program.
<code>gGuiwin()</code>	This is the device nomination routine for running GINOMENU under Windows.
<code>gmInitializeMenu()</code>	This initializes GINOMENU
GINOMENU definition routines	GINOMENU calls to define widgets
<code>gmManage()</code>	Start GUI management
<code>gmAction(callback)</code>	Start GUI Action loop
Application Code	Code to perform as a result of user interaction, before returning back to Action loop
<code>gCloseMenu()</code>	This call closes down GINOMENU and frees up any used resources.
<code>gCloseGino()</code>	This call closes down GINO. It includes an implicit call to <code>gCloseDevice()</code> to ensure a tidy shutdown.

## The Structure of a simple non-GINOMENU GUI program under Windows

A GUI program developed under Delphi, VB or .NET has to initialize GINO and an output window, then all GINO graphics commands are placed in the event-handling routines until program termination whereby GINO is closed down in the usual manner, however the graphics window also needs to be updated on a Paint event to keep the graphics up to date:

## Delphi Code

```
procedure TForm1.FormCreate(Sender: TObject);
var
  myHdc: Integer;

begin
  myHdc:=Integer(Image1.Canvas.Handle);
  gOpenGino;
  gMwindc(GNULL, myHdc);
end;

procedure TForm1.FormPaint(Sender: TObject);
var
  mynewHdc: Integer;

begin
  mynewHdc:=Integer(Image1.Canvas.Handle);
  gFlushGraphics;
  gMwinUpdateDC(mynewHdc);
end;

procedure TForm1.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  gCloseGino;
end;
```

---

## GINO Coordinate System

The default coordinate system in GINO is millimetres. GINO was designed to produce accurately scaled drawings on any device it was outputting to, therefore the call

```
gDrawLineTo2D(100.0,0.0)
```

will produce a line exactly 100.0 millimetres long on a printer or plotter and 100.0 millimetres long on a screen (bearing in mind the difficulty in measuring lines on a screen!).

To change GINO's default coordinate system to inches, use the following call at the beginning of a program:

```
gDefinePictureUnits(25.4)
```

If a 'scale-to-fit' mapping of a drawing to the output device is required and exact measurements are not important, then a GINO viewport can be set up using the routine `gSetViewport2D()`. For example if your drawing has notional dimensions of 1000.0 x 750.0, then the following segment of code will enquire the drawing dimensions of the output device and automatically map the 1000.0 x 750.0 to these dimensions:

## C Code

```

GDIM    paper;
static GLIMIT window = {0.0, 1000.0, 0.0, 750.0},
        viewport;

gOpenGino();
xxxxx();
gEnqDrawingLimits(&paper, &ipapty);
viewport.xmin = 0.0;
viewport.xmax = paper.xpap;
viewport.ymin = 0.0;
viewport.ymax = paper.ypap;
gSetViewport2D(&window, &viewport);

```

## F90 Code

```

type (GDIM) :: paper
type (GLIMIT) :: picture = GLIMIT(0.0,1000.0,0.0,750.0)
type (GLIMIT) :: viewport
!
call gOpenGino
call xxxxx
call gEnqDrawingLimits (paper, ipapty)
viewport%xmin=0.0
viewport%xmax=paper%xpap
viewport%ymin=0.0
viewport%ymax=paper%ypap
call qSetViewport2D(picture,viewport)

```

## Origin of Coordinate System

The origin for all GINO's vector drawing is in the bottom left-hand corner and the origin for GINO's pixel output is in the top left-hand corner.

GINOMENU's origin is compatible with GINO and retains a bottom left-hand origin for all widget placement and drawing within graphics frames, however, GINOMENU does have the ability to change its origin either for all widgets or selectively.

---

## GINO Configuration File

When GINO is initialized, one of the firsts steps carried out by the library is to check for the existence of a legal Configuration File. If this does not exist or does not have the correct licence information encoded within it, GINO will immediately stop with an appropriate error message. In order to provide a user-controllable check on the Configuration File, the GINO initialization routine `gOpenGino()` can be replaced with `gEnqConfigStatus()` as described in the GINO User's Guide Introduction. This function can also be used to specify the location of the GINO configuration file to save having to rely on environment variables when installing end-user applications.

---

## Windows

Under Windows, the GINO configuration file is called GINO.CON and must be located in one of the following directories:

Current Directory

Directory pointed to by the environment variable GINO

Directory pointed to by the routine gEnqConfigStatus()

\GINO (on the current drive)

## UNIX and Linux

Under UNIX/Linux, the GINO configuration file is called gino.config and must be located in one of the following directories:

Current Directory

Directory pointed to by the environment variable GINOHOME

Directory pointed to by the routine gEnqConfigStatus()

/usr/local/lib

/usr/local

## OpenVMS

Under OpenVMS, the GINO configuration file is called GINO.CON and must be located in one of the following directories:

Current Directory

Directory pointed to by the logical name GINOHOME

Directory pointed to by the routine gEnqConfigStatus()

Remember that when distributing GINO programs, the GINO configuration file must also be distributed and must also be located in a valid directory.

The GINO configuration file may also contain system administration commands to set or alter system or device parameters specific to the local implementation. This includes Error output, Window Titles, Window Colours, Fortran output channels etc. All these settings are described in the GINO User Guide Appendix A and B.

---

## Internal Naming Convention

When creating routines that call GINO subroutines, it is important that users do not choose names that will clash with internal names used by GINO.

The names of routines internal to the GINO libraries are too numerous to list, but they adhere to a strict convention. All names take one of the following three forms:

GINOnn	GINO internal names
GFnnnn	GINO internal names
GGnnnn	GINOGRAF internal names
GSnnnn	GINOSURF internal names
GMnnnn	GINOMENU internal names

where n can be any alphanumeric character.

---

## Error Handling

All GINO libraries generate error and warning messages if any errors or inconsistencies are found and the list of messages are found in the Appendices of each manual.

An error message is generated if GINO is unable to perform an operation. For instance, a message is generated if GINO is asked to output when no output device has been nominated. Another example is that the routine `ggPlotBarChart()` generates a GINOGRAF Error 1 if the number of bars in the Bar Chart is negative or zero.

A warning message is generated if GINO is able to execute a command by assuming default information, although illegal information has been supplied. For example, the routine `ggSetPieChartExplosion()` generates a GINOGRAF Warning 24 if a segment explosion factor is negative. In such cases the routine takes the absolute value or a default value, and continues.

Error and warning messages consist of the error or warning number and on most implementations the text of the associated message as found in the appendices. In order to assist debugging, the routine in which the error or warning was detected is also printed and if a GINO error was generated by a call from another GINO routine, both routine names are given. For example:

```
GINO Warning 56 - Line style index out of range
Detected in gSelectLineStyle from ggDisplayLineColumn
```

By default, error messages are sent to either the command window that initiated the program, in the case of running under UNIX or OpenVMS, or a new error window is created if running under Microsoft Windows.

The user can alter the initial state of this error window through the configuration variable NFERTR set in the configuration file GINO.CON as follows:

NFERTR	
= 0	Visible (Default)
= 1	Iconized
= 2	Hidden
> 2	Errors are sent to a file called "application".err (e.g. myprog.err)

The error channel and filename can be changed using GINO's error handling routines as described in the GINO User's Guide Introduction.

Other diagnostic facilities provided in GINO are the 'tracer' feature which outputs a message every time a GINO routine is called and the DEBUG pre-processor which outputs a log of all internal calls made to the device-driver. Both of these are also described in the GINO User's Guide Introduction.

---

## Redistributing GINO files

### PC Redistribution

The GINO licence on a PC includes an automatic unlimited run-time licence allowing you to distribute your GINO executable programs free of charge. Certain GINO files are required for correct operation of your program and these are described below:

WINGUIxx.DLL	Always required
GMWINERR.DLL	Always required
GMENUDLL.DLL	Always required
GWOGL.DLL	Always required
GINO.CON	Always required
GINLIBxx.DLL	Only required if application uses GINO-F
GMWIN.DLL	Only required if application uses GINO-F
GJPEG.DLL	Only required if application uses GINO-F
GPNG.DLL	Only required if application uses GINO-F

---

GGRAFxx.DLL	Only required if application uses GINOGRAF
GSURFxx.DLL	Only required if application uses GINOSURF
SYSMODxx.DLL	Only required by 95 applications using GINO-F

(If distributing double-precision applications, the above files with 'D' appended to them are distributed instead).

GINO.FON	Only required if using GINO software fonts above font no. 0
GINO.MES	Only required if application is to generate long error text (error numbers will still be generated without this file)

All the above files either need to be placed in the same directory as the application or the DLL's can be placed in the system PATH and the .CON, .FON and .MES files can be placed in a directory pointed to by the environment variable GINO.

Some compilers may need their own DLL's distributed with the application - check the compiler documentation for details.

If distributing applications to PCs running Win95 Release 1, the OpenGL DLL's OPENGL32.DLL and GLU32.DLL found in \WINDOWS\SYSTEM are required. These are required even if your GINO application is not making use of OpenGL functionality.

Win95 Release 2, Win98, NT, 2000 and XP are all supplied with these DLL's automatically.

## UNIX, Linux and OpenVMS Redistribution

To redistribute GINO programs on UNIX/OpenVMS platforms a run-time licence is required from Bradly Associates. All programs on these platforms are statically linked, so only the following GINO files are required:

### UNIX/Linux

gino.config	Contains your serial no. and is always required
gino.fonts	Only required if using GINO software fonts above font no. 0
gino.message	Only required if application is to generate long error text (error numbers will still be generated without this file)

---

MENUBIG1.ICN	Required for GINOMENU programs using GINOMENU's large supplied icons
MENUSML1.ICN	Required for GINOMENU programs using GINOMENU's small supplied icons

**OpenVMS**

GINO.CON	Contains your serial no. and is always required
GINO.FON	Only required if using GINO software fonts above font no. 0
GINO.MES	Only required if application is to generate long error text (error numbers will still be generated without this file)
MENUBIG1.ICN	Required for GINOMENU programs using GINOMENU's large supplied icons
MENUSML1.ICN	Required for GINOMENU programs using GINOMENU's small supplied icons

These files need to be placed either in the same directory as the application or in a directory pointed to by the environment variable GINOHOME.

# Index

---

---

## C

C/C++ version ······ 12 - 14  
Configuration file ······ 25

---

## D

Delphi ······ 15  
Derived types ······ 10  
Double-precision ······ 7,26

---

## E

Error handling ······ 24  
Example Programs ······ 7,12

---

## F

F77 - Porting from ······ 8  
F90 Version ······ 7 - 11

---

## G

GINO.CON ······ 22  
GINOGRAF ······ 5  
GINOMENU ······ 6  
GINOMENU-X ······ 17  
GINOSURF ······ 5  
GUI programs ······ 18

---

## I

Inches ······ 21  
Include Files ······ 13  
Internal subroutine names ······ 23

---

## L

Linux version ······ 17,26  
Long descriptive names ······ 9

---

## M

Mapping coordinates ······ 21  
Millimetres ······ 21  
Module Files ······ 9

---

## N

Non-GUI programs ······ 18

---

## O

OpenGL ······ 5  
OpenVMS version ······ 17,26  
Optional arguments - F90 ······ 11  
Origin ······ 22

---

## P

Pre-defined constants ······ 10  
Printing ······ 19

---

## R

Run-time licencing ······ 25 - 27

---

## S

SCALE routine ······ 12  
Structures - C/C++ ······ 14  
Structures - F90 ······ 10

---

## U

UNIX version ······ 17,26

---

## V

Variable Arguments - C/C++ ······ 14  
Viewport ······ 21  
Visual Basic ······ 16

---

---

Visual Studio .NET . . . . . 16