Table of Contents

Introduction
- In This Introduction .................................................. 3
- About This Manual .................................................. 3
  - Organization of This Manual ........................................ 3
  - Types of Users .................................................. 4
  - Software Dependencies .......................................... 5
- Documentation Conventions ........................................ 5
  - Typographical Conventions .................................... 5
  - Icon Conventions ............................................. 6
- Additional Documentation ......................................... 7
- Informix Welcomes Your Comments .............................. 7

Section I Installing Dynamic 4GL

Chapter 1 Installing Dynamic 4GL
- In This Chapter ..................................................... 1-3
- UNIX Installation ................................................... 1-4
  - Prerequisites for UNIX ........................................ 1-4
  - Automatic UNIX Installation ................................ 1-5
  - Manual UNIX Installation ..................................... 1-9
  - Post-Installation Tasks ....................................... 1-11
- Installing Web Deployment Components on UNIX .......... 1-14
- Installation Directory Tree ..................................... 1-24
- Upgrading from Earlier Software Versions .................. 1-26
- Microsoft Windows NT Installation ............................ 1-26
  - Prerequisites for Microsoft Windows NT Systems .......... 1-27
  - Step-by-Step Installation on Windows NT .................. 1-28
  - NT Configuration .............................................. 1-31
  - Installing Web Deployment Components on NT ............ 1-53
Chapter 2  Configuring Your Client Environment

In This Chapter ........................................... 2-3
UNIX Front-End Configuration .......................... 2-3
   Installing the X11 Front End ....................... 2-4
   Managing Application Windowing ................. 2-5
Windows Front-End Configuration ...................... 2-8
   Installing the Windows Front End ............... 2-8
   How to Use the Windows Front End .............. 2-9

Section II  Using Dynamic 4GL

Chapter 3  Overview of Dynamic 4GL

In This Chapter ........................................... 3-3
About Dynamic 4GL ...................................... 3-3
Basic Principles ......................................... 3-4
   Software Architecture ............................... 3-4
   P-Code and C Code ................................... 3-4
Compilation Tools ...................................... 3-5
   Main Compilation Tools ............................. 3-5
   Other Compilation Tools ......................... 3-6
   Configuration Tools ................................. 3-6
   Miscellaneous Programs and Scripts .......... 3-7
Required Environment Variables ..................... 3-7
Compiling a Simple Program .......................... 3-8
   Setting up the Environment ...................... 3-9
   Writing the Source Code ......................... 3-10
   Compiling the Source Code ..................... 3-11
   Compiling the Form File ......................... 3-12
Running the Program on the Front End .............. 3-13

Chapter 4  Using the Compiler

In This Chapter ........................................... 4-3
Compiling Forms and Help Messages ................. 4-3
   Compiling Form Specification Files .............. 4-3
   Compiling Help Message Files .................. 4-4
   Generating a New Database Schema File ........ 4-4
Compiling to P-Code .................................. 4-5
   Overview of the P-Code Example ............... 4-5
   Using C Functions in 4GL Applications Compiled to P-Code . 4-7
Chapter 5 Differences Between 4GL and Dynamic 4GL Compilers
In This Chapter .......................... 5-3
Differences Between 4GL and Dynamic 4GL Compilers .... 5-3
The Dynamic 4GL Configuration Files ................... 5-3
List of Differences .......................... 5-4
4GL Extensions ............................ 5-6

Chapter 6 Extensions to the 4GL Language
In This Chapter .......................... 6-5
Non-Graphical 4GL Extensions .................... 6-5
Channels: Handling Input/Output Streams ............ 6-5
DDE Communication Protocol .................... 6-10
Extension of DISPLAY ARRAY Command ............ 6-15
Other Non-Graphical Extensions .................. 6-16
New Form Specification Fields .................... 6-23
List Boxes ................................ 6-23
Buttons ................................... 6-25
Bitmaps ................................... 6-29
Check Boxes and Radio Buttons ................. 6-30
Combo Boxes ............................... 6-33
Scrolling Fields ............................ 6-34
Graphical 4GL Extensions ...................... 6-35
General Display Extensions .................... 6-35
Dialog Boxes ................................ 6-39
New Input Statement Functions ................. 6-44
4GL Window Management Functions ............ 6-54
Extensions for Drawing in 4GL .................. 6-59
Defining the Drawing Area .................... 6-59
Extensions for Drawing Shapes .................. 6-60
Toolbars ................................. 6-66
Appendix A  Environment Variables
Appendix B  The Configuration File
Appendix C  Common Problems and Workarounds
Appendix D  The Character Filter: mkchartab
Appendix E  Configuration Manager
          Error Messages
          Index
## Introduction

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>In This Introduction</td>
<td>3</td>
</tr>
<tr>
<td>About This Manual</td>
<td>3</td>
</tr>
<tr>
<td>Organization of This Manual</td>
<td>3</td>
</tr>
<tr>
<td>Types of Users</td>
<td>4</td>
</tr>
<tr>
<td>Software Dependencies</td>
<td>5</td>
</tr>
<tr>
<td>Documentation Conventions</td>
<td>5</td>
</tr>
<tr>
<td>Typographical Conventions</td>
<td>5</td>
</tr>
<tr>
<td>Icon Conventions</td>
<td>6</td>
</tr>
<tr>
<td>Additional Documentation</td>
<td>7</td>
</tr>
<tr>
<td>Informix Welcomes Your Comments</td>
<td>7</td>
</tr>
</tbody>
</table>
In This Introduction

This introduction provides an overview of the information in this manual and describes the conventions used.

About This Manual

The Informix Dynamic 4GL User Guide describes how to develop 4GL applications on UNIX and then deploy them on various platforms, such as Windows 95, NT, and Motif.

This manual assumes that you already have a complete set of INFORMIX-4GL manuals, such as the INFORMIX-4GL Reference manual, and the appropriate SQL manuals. This manual should be used in addition to the 4GL manuals.

Organization of This Manual

This manual includes the following chapters:

- Chapter 1, “Installing Dynamic 4GL,” describes how to install this product on either UNIX or Windows.
- Chapter 2, “Configuring Your Client Environment,” describes how to configure your system on both UNIX and Windows.
- Chapter 3, “Overview of Dynamic 4GL,” provides a quick overview of Informix Dynamic 4GL concepts, a description of the tools, the required environment variables, and the procedure for compiling a simple program.
- Chapter 4, “Using the Compiler,” describes how to compile various types of files, and how to compile to p-code or C code.
Types of Users

This manual is written for all Informix Dynamic 4GL users.

This manual is written with the assumption that you have the following background:

- A thorough knowledge of INFORMIX-4GL
- Some experience working with relational databases or exposure to database concepts
Software Dependencies

This manual is written with the assumption that you are using a supported Informix server.

Documentation Conventions

This section describes the conventions that this manual uses. These conventions apply to all Informix documentation.

The following conventions are covered:

- Typographical conventions
- Icon conventions

Typographical Conventions

This manual uses the following conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYWORD</td>
<td>All primary elements in a programming language statement (keywords) appear in uppercase letters in a serif font.</td>
</tr>
<tr>
<td><em>italics</em></td>
<td>Within text, new terms and emphasized words appear in italics.</td>
</tr>
<tr>
<td><em>italics</em></td>
<td>Within syntax and code examples, variable values that you are to specify appear in italics.</td>
</tr>
<tr>
<td><em>boldface</em></td>
<td>Names of program entities (such as classes, events, and tables), environment variables, file and pathnames, and interface elements (such as icons, menu items, and buttons) appear in boldface.</td>
</tr>
</tbody>
</table>
Icon Conventions

When you are instructed to “enter” characters or to “execute” a command, immediately press RETURN after the entry. When you are instructed to “type” the text or to “press” other keys, no RETURN is required.

**Icon Conventions**

Comment icons identify three types of information, as the following table describes. This information always appears in italics.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="warning_icon.png" alt="Warning" /></td>
<td>Warning:</td>
<td>Identifies paragraphs that contain vital instructions, cautions, or critical information</td>
</tr>
<tr>
<td><img src="important_icon.png" alt="Important" /></td>
<td>Important:</td>
<td>Identifies paragraphs that contain significant information about the feature or operation that is being described</td>
</tr>
<tr>
<td><img src="tip_icon.png" alt="Tip" /></td>
<td>Tip:</td>
<td>Identifies paragraphs that offer additional details or shortcuts for the functionality that is being described</td>
</tr>
</tbody>
</table>
Additional Documentation

Informix Dynamic 4GL documentation is provided in a variety of formats:

- **Printed documentation.** The *Informix Dynamic 4GL User Guide* is available as a printed manual.
- **On-line manuals.** You have the ability to print chapters or entire books and do full-text searches for information in specific books or throughout the documentation set. On-line manuals are available through Answers OnLine. You can order Answers OnLine on a CD, or if you have access to the Web, visit the following URL: www.informix.com/answers.
- **Release notes.** Release notes are located in the */release* directory where the product is installed. Please examine these files because they contain vital information about application and performance issues.
- **HTML files.** The HTML client, which lets you deploy applications on a Web server, is documented in HTML files. See Chapter 1 for more information.

Informix Welcomes Your Comments

Let us know what you like or dislike about our manuals. To help us with future versions of our manuals, we want to know about any corrections or clarifications that you would find useful. Include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

Write to us at the following address:

Informix Software, Inc.
Tools Technical Publications Department
4100 Bohannon Drive
Menlo Park, CA 94025
Informix Welcomes Your Comments

If you prefer to send electronic mail, our address is:

doc@informix.com

We appreciate your suggestions.
Section I

Installing Dynamic 4GL

Chapter 1  Installing Dynamic 4GL
Chapter 2  Configuring Your Client Environment
Chapter 1

Installing Dynamic 4GL

In This Chapter ............................................. 1-3
UNIX Installation ............................................. 1-4
    Prerequisites for UNIX .................................. 1-4
        C Prerequisites ....................................... 1-4
        Prerequisites for the Development Server .......... 1-4
        TCP/IP Prerequisites .................................. 1-5
        Disk Space Prerequisites ............................... 1-5
Automatic UNIX Installation ............................. 1-5
    Installation from CD ..................................... 1-6
    Installation Without CD: The Self Extracting Archive File .... 1-6
    Installation Process ..................................... 1-6
    License Installation ..................................... 1-7
Manual UNIX Installation ................................. 1-9
    Logging On and Loading the Files ..................... 1-9
    Manual Installation Process ............................ 1-9
    Manual License Installation ............................ 1-10
Post-Installation Tasks ................................. 1-11
    The C Compiler ......................................... 1-11
    Finding the Required Libraries: findlib.sh ............ 1-12
    Creating the P-Code Runner and Libraries ............. 1-12
    Creating the C Code Libraries .......................... 1-14
Installing Web Deployment Components on UNIX ......... 1-14
    Prerequisites for Installing the Web Deployment Components . 1-14
Important Terms ........................................... 1-15
Components on the CD ...................................... 1-15
Automatic Installation ................................. 1-15
Manual Installation ...................................... 1-18
Troubleshooting the UNIX Installation ................. 1-22
Installation Directory Tree .............................. 1-24
Upgrading from Earlier Software Versions .............. 1-26
In This Chapter

Two different packages are distributed: the development package and the runtime package. The development package includes all the tools needed to compile and execute your 4GL programs. The runtime package is a lighter version of Dynamic 4GL. It allows you to execute previously compiled programs, but it does not allow you to compile them. There is a runtime package for each version of the development package. The installation process is the same for both.

Be sure to read Chapter 2, “Configuring Your Client Environment,” before doing your first installation of Dynamic 4GL.

**Important:** Once you license the product, you cannot change the directory where the product is installed.

You can also install components to deploy your Dynamic 4GL applications on a Web server, allowing users to access the application from a Web browser. Complete documentation for using this option is available in HTML files when you install the Web deployment components, in the CLIENTS/CLI-HTML directory on UNIX systems or the \Clients\CLI-HTML directory on NT systems. Instructions for installing these components are provided later in this section for both UNIX and NT. For UNIX, see “Installing Web Deployment Components on UNIX” on page 1-14. For NT, see “Installing Web Deployment Components on NT” on page 1-53.
UNIX Installation

This section describes the prerequisites for installation, the installation process itself, and the directory structure created by the installation.

Prerequisites for UNIX

Before executing the installation program, be certain that your system meets the following requirements.

C Prerequisites

When you install the development version of Dynamic 4GL, you need to link your system libraries, your INFORMIX-ESQL/C libraries, and the Dynamic 4GL libraries. To perform this linking, you need a C compiler on your development system. As a result of this link, the runner is created. You need to relink each time one of these three components changes: your operating system, your database interface, or the version of Dynamic 4GL.

You only need an ANSI C compiler in two cases:

- If you call C functions within your program
- If you want to compile your applications to C code and not to p-code

If you want to use your native C compiler instead of the GNU C compiler delivered with Dynamic 4GL, check that it conforms to ANSI standards. It must also accept the `-c` flag, to produce object files, and the `-o` flag, to produce executable files.

See the section “The C Compiler” on page 1-11 for instructions on how to install the GNU C compiler from the Dynamic 4GL distribution CD.

Prerequisites for the Development Server

For technical and legal reasons regarding Informix products, you should have at least one INFORMIX-ESQL/C development server installed to compile programs to C code or to create p-code runners and to be allowed to resell them.
Automatic UNIX Installation

Note that if you do not have INFORMIX-ESQL/C or INFORMIX-4GL (compiled) installed, the installation program will install a version without the database interface (fglnodb runner). In this case, the program will generate errors if you try to execute SQL statements in your 4GL programs.

TCP/IP Prerequisites

A development version of TCP/IP must exist on the development server. Check your operating system manuals to be sure it is installed as the default. As an example, on SCO systems 3.2, the library libsocket.a must be present.

Disk Space Prerequisites

The installation requires some space in the /tmp directory: approximately 15 MB for working space, which is released when the installation process is complete, and approximately 20 MB for the compiler and the Tcl/Tk graphical display server, which will be installed in the file system or systems you choose.

You can install the display server in the same directory as the Dynamic 4GL compiler, or you can separate the two. If you expect to install a compiler license and a runtime license for Dynamic 4GL on the same machine, it is best to keep the display server code separate.

Automatic UNIX Installation

Dynamic 4GL provides an installation program to make the installation process easier. This installation program uses Bourne shell scripts executable by all UNIX systems. The following steps might be performed by the installation process:

- Back up a previous version of Dynamic 4GL if a version is already installed on your system.
- Install all the needed files on your system.
- Create a default environment for compilation.
- License the compiler if needed or keep the previous license for an update.
- Create the p-code runner and all the needed libraries and tools.
Automatic UNIX Installation

**Installation from CD**

If you have a CD accessible from your UNIX system, log on as the system super user, mount the Dynamic 4GL CD, and go into the mount directory. Then run the shell script named `install.sh` with the following command:

```
$ /bin/sh ./install.sh product_type
```

where *product type* is the package you want to install. The following table lists the packages available.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compiler</td>
<td>Installs a compiler</td>
</tr>
<tr>
<td>runtime</td>
<td>Installs a runtime package</td>
</tr>
<tr>
<td>patch</td>
<td>Installs a patch over a version</td>
</tr>
<tr>
<td>demo</td>
<td>Installs the trial package</td>
</tr>
</tbody>
</table>

**Installation Without CD: The Self Extracting Archive File**

If you do not have a CD accessible from your UNIX machine, just copy the file located in the directory `/OS/UNIX/your_OS_name/SLF/` corresponding to the package you plan to install in your UNIX system.

Then log on, go into the directory where you have copied the package, and run the following command if you want to install a development package:

```
$ /bin/sh ./OSType.sh -i
```

This shell creates all the necessary files and starts the installation process.

**Installation Process**

The installation process first attempts to determine your host operating system, and checks if all the system requirements are met. It then copies the product into a temporary directory and searches for any existing Informix and Dynamic 4GL products to set the `INFORMIXDIR` and `FGLDIR` environment variables. If you want to overwrite an installed version of the compiler with a new version, you will be prompted for the creation of a backup archive of the existing compiler.
Automatic UNIX Installation

After these operations, the install shell copies the Dynamic 4GL files to the specified install directory.

The next step of the installation process is to determine the needed system and Informix libraries to compile 4GL applications. This is done by the `findlib.sh` Bourne shell script located in the `/bin` subdirectory of the installation directory. Read “Finding the Required Libraries: findlib.sh” on page 1-12 for more information about this shell script.

During the installation process, the program will prompt for a license. See the next section, “License Installation,” for a step-by-step procedure to install your license successfully.

After the license is successfully installed, the installation shell script generates the p-code and the C code 4GL libraries. A Bourne shell script is also generated in the installation directory. This shell script is named `envf4gl`, and it sets up the main environment variables required for using Dynamic 4GL. Informix recommends including a call to this shell script in your session startup file (.login or .profile on most UNIX systems).

License Installation

Dynamic 4GL licenses are made of the following numbers:

- Serial number
- Serial number key
- Installation number
- Installation key

The serial number and serial number key are provided with the Dynamic 4GL media. The installation number is generated for you. After the installation number is generated, you will need to access the installation key by going to the following Web site: www.informix.com/keyissue.

The serial number follows the standard Informix format—an 11-character alphanumeric string. The remaining numbers and keys are always built on the same architecture: 12 uppercase letters and digits followed by an optional checksum number. In order to avoid any confusion, the letter O is never used; it is always the digit 0 (zero).
Before you start the licensing process, the following message is displayed:

```
INFORMIX license manager program
This program must be run as root or by the user installing the software.
Press RETURN to continue, or the interrupt key (CTRL-C or DEL) to abort.
```

Press RETURN to start the licensing process. (You can also press the system interruption key to interrupt the process and redisplay the UNIX command prompt.)

Then you will be prompted for the serial number. Type it and press RETURN; for example, FJD#D253864:

```
Enter your serial number (e.g., XXX#XAAAAAA) > FJD#D253864
```

The next step is to enter the serial number key. After this operation, the installation number is generated. This number allows you to get the installation key.

Use the on-line licensing page on the Informix Web server (www.informix.com/keyissue) to access the installation key. You have 30 days to enter this key. If you must enter the key at a later date, use the following command to complete the license installation:

```
$ licencef4GL -k serial_number_key
```

**Tip:** Never use the letter O but always the digit 0 (zero), except for the check numbers.

If you have to reinstall the software from a backup, to avoid having to get a new serial number key, you must reinstall the product in the same location, either physically or using a logical link to the same FGLDIR directory.

If you have to change your serial number, you must first uninstall the current license. To do so, run:

```
$ fglWrt -d```
Manual UNIX Installation

Although you can install Dynamic 4GL software using the installation program, Dynamic 4GL also allows you to install the product manually.

Logging On and Loading the Files

Log on as root. If you have an earlier version of the software on your system, make sure no one is using it during the installation of the new one, and stop all Dynamic 4GL daemons.

Files with the extension .tgz are compressed archive files. To uncompress this kind of file, you must first run gzip and then tar with the following commands:

```
$ gzip -d filename.tgz
$ tar xvf filename.tar
```

Before decompressing the file with tar, you can view its contents with the following command:

```
$ tar tvf filename.tar
```

Distributions on tapes can be loaded with the commands:

```
$ cd /tmp
$ tar xvf /dev/your_device
```

On the distribution CD, you will find all the necessary files in the /OS/UNIX/your OS name directory.

Manual Installation Process

The archive file is assumed to be named:

```
F4GL.TGZ
```

This archive file contains a complete directory tree, which can be installed anywhere.
A convenient way to proceed is:

```bash
$ mkdir installdir/f4gl.version
$ cd installdir/f4gl.version
$ FGLDIR=installdir/f4gl.version
$ export FGLDIR
$ INFORMIXDIR=Path_to_Informix_directory
$ export INFORMIXDIR
$ cp path_to_gzip_file .
$ gzip -d F4GL.TGZ
$ tar xvf F4GL.tar
```

where `installdir` is the path to the installation directory. The following examples illustrate how to do both a new installation and an update.

For a new installation:

```bash
$ mkdir /usr/f4gl
$ cd /usr/f4gl
$ FGLDIR=/usr/f4gl
$ export FGLDIR
$ INFORMIXDIR=/usr/informix4.1
$ export INFORMIXDIR
$ cp CD/OS/UNIX/SCO/COMPILER/F4GL.TGZ .
$ gzip -d F4GL.TGZ
$ tar xvf F4GL.tar
```

For an update, first make a backup of your earlier version:

```bash
$ mkdir /usr/f4gl.save
$ cd /usr/f4gl
$ tar cvf /usr/f4gl.save/f4gl-version.tar .
$ gzip /usr/f4gl.save/f4gl-version.tar
```

Now you can install the new one:

```bash
$ cp CD/OS/UNIX/SCO/COMPILER/F4GL.TGZ .
$ gzip -d F4GL.TGZ
$ tar xvf F4GL.tar
```

**Manual License Installation**

To install or reinstall a license, the `FGLDIR` environment variable has to be set to the directory where you have installed the product, and the `$FGLDIR/bin` directory has to be added to your `PATH` variable. Then execute the following commands:

```bash
$ cd $FGLDIR/bin
$ licencef4gl
```
This will start the license installation process, as described in “License Installation” on page 1-7.

Post-Installation Tasks

If you are doing a manual installation, you will need to complete the following procedures by hand before you will be able to use Dynamic 4GL. If you are performing an automatic installation, these tasks are done for you.

The C Compiler

During this phase, you might need a C compiler. It is required if you plan to create a new runner or if you want to compile your 4GL programs to C code. But it is not used afterward for p-code compilation. You can use either the native C compiler of the machine or the C compiler of the GNU tools, included on the distribution media.

If there is no usable native C compiler on your machine, the GNU tools must be installed. Note that you still need to have your UNIX system libraries installed.

To install the GNU C compiler from the Dynamic 4GL CD, go into the OS/UNIX/ directory and run the following command:

```
$ /bin/sh ./insttgcc -i
```

This shell will install the package GCC.TGZ located in the directory /OS/UNIX/your_OS_name/GNUC.

If you cannot mount the CD directly on your UNIX machine, you can copy the file gnuc.sh, located in the directory /OS/UNIX/your_OS_name/SELFEXTR, to a temporary directory on your UNIX machine. Use binary transfer mode because this shell script contains all the files of the GNU C compiler. Then run the following command to start the installation:

```
$ /bin/sh ./GNUC.SH -i
```

During the installation process, you will be prompted for the installation directory of the GNU C compiler. A shell script named envgcc will also be generated during the installation. You must execute this shell to set all the environment variables needed for compiling and linking C programs.
Post-Installation Tasks

**Tip:** This distribution does not contain the system libraries you need to compile C sources. To obtain those libraries, contact your operating system reseller.

If you plan to link a runner without any C functions, you only need to install a linker, and not an ANSI-C compliant compiler.

If you are not using the default C compiler (normally `cc`), make sure that you have set the `INFORMIXC` environment variable to the compiler you are using (such as `gcc` for the GNU C compiler) as well as the documented `FGLCC` and `CC` variables. For example:

```
INFORMIXC=gcc
export INFORMIXC
```

**Finding the Required Libraries: findlib.sh**

The first step is to identify the Informix libraries, the UNIX system libraries, and the Dynamic 4GL libraries needed to create the libraries and the p-code runner. To do so, run the `findlib.sh` Bourne shell script located in the `bin` subdirectory where Dynamic4GL is installed. This step requires a C compiler and the INFORMIX-ESQL/C development libraries:

```
$ /bin/sh ./findlib.sh
```

This script generates a file called `envf4gl` in the local directory. This shell script sets all the environment variables necessary to create the p-code runner and the 4GL libraries, which allow you to compile to C code and to execute 4GL programs. Execute this Bourne shell script with the following command:

```
$ ./envf4gl
```

**Creating the P-Code Runner and Libraries**

You are now ready to create the p-code runner. This runner contains all the routines to access to the Informix database with your version of the Informix database interface. This runner is used when you link your 4GL source code modules together and when you run the p-code compiled 4GL programs.
The runner is the result of linking your Informix libraries, your UNIX system libraries, and the Dynamic 4GL libraries. Each time that one of these three components changes, you must create a new runner. If you have C functions, you must also include them in the runner. For more information about using C functions with 4GL, see “Using C Functions in 4GL Applications Compiled to P-Code” on page 4-7.

**Important:** Creating the p-code runner for your machine requires a C compiler and the INFORMIX-ESQL/C development libraries.

If you have Version 4.x or 5.x Informix database servers, use the following syntax:

```
$ fglmkrun
```

If you have Version 6.x or 7.x Informix database servers, set the FGLDBS environment variable to `ix711`, or use the `-d ix711` option with the `fglmkrun` tool to create the runner:

```
$ fglmkrun -d ix711
```

This command creates the default p-code runner, called `fglrn`, in the `$FGLDIR/bin` directory.

Note that with SCO systems, the use of `fglmkrun` during a manual installation may cause the following error message:

```
Symbol not found
First referenced in file
fileno .../lib/libf2c.a
```

The solution is to first create a file named `fileno.c`, containing the following code:

```c
#include stdio.h
#undef fileno
int fileno(f)
FILE *f ;
{
return(f->__file) ;
}
```

Next, execute `fglmkrun` with `fileno.c` as an additional parameter (for Informix Version 5.x):

```
$ fglmkrun -o fglrun fileno.c $FGLDIR/lib/fglExt.c
```

This creates the runner named `fglrn` in the current directory.
Installing Web Deployment Components on UNIX

After you have successfully created the p-code runner, run the `rtsinstall` command to create the p-code libraries and tools:

```
$ rtsinstall
```

**Creating the C Code Libraries**

If you have Version 6.x or 7.x Informix database servers, set the `FGLDBS` environment variable with `ix711`:

```
$ FGLDBS=ix711
$ export FGLDBS
```

Then run the `fglinstall` program in order to compile the C code libraries and tools:

```
$ fglinstall
```

You are now ready to compile 4GL programs on your UNIX system.

Installing Web Deployment Components on UNIX

This section contains instructions for installing the Web deployment components of Dynamic 4GL on your UNIX system.

**Prerequisites for Installing the Web Deployment Components**

To install this software, you need the following:

- For the UNIX install, the `gzip` and the `tar` utilities
- A running Dynamic 4GL compiler with a valid license number
- A running Web (HTTP) server
- A browser with HTML form and table support (HTML, Version 3.2 or higher).
  
  These features are present in Microsoft Internet Explorer, Version 2.x or higher, and in Netscape Navigator, Version 2.x or higher.
Installing Web Deployment Components on UNIX

Important Terms

Because Web deployment implies a client-server configuration, the following terms are important to understand before you proceed:

- The application server runs the main Dynamic 4GL program.
- The HTML client, fglcl, handles communication with the HTML server.
- The HTML server, fglhtmld, handles and controls the HTML client’s runner.
- The Web server is the location where the Web server daemon, named httpd or ns-httpd, is running.

As with the basic Dynamic 4GL software, you can install the Web deployment components either through a script (automatic installation) or manually. The rest of this section describes the two installation processes.

Components on the CD

Once you have completed installing the Dynamic 4GL software, the components for Web deployment are available in the CLIENTS/CLI-HTML directory on the CD. The following subdirectories are present:

```
ALL
BIN
DOC
SELFEXTR
TLB
```

In addition, you will find the installation script, install.sh.

Automatic Installation

You can begin the automatic installation either by executing a self-extracting package or by executing a Bourne shell script. Once you have executed either of these, you can proceed to answer a series of prompts that appear.

Using the Self-Extracting Package

To install all the Web deployment components, enter:

```
sh html-all.sh -i
```
Installing Web Deployment Components on UNIX

To install only the binaries and the example, enter:

```
sh html-bin.sh -i
```

To install only the documentation, enter:

```
sh html-doc.sh -i
```

Binaries are included for all supported UNIX systems.

**Using the Shell Script**

First, extract the files. If you have the GNU version of the `tar` program, enter:

```
tar -xzf HTML.tgz
```

If you do not have the GNU version of `tar`, enter:

```
gunzip -c HTML.tgz | tar -xzf -
```

Next, run the script. To install the complete package, enter:

```
sh install.sh all
```

To install only the binaries and the example, enter:

```
sh install.sh binary
```

To install only the documentation, enter:

```
sh install.sh doc
```

To access on-line help, enter:

```
sh install.sh -h
```

**Responding to the Prompts**

For the complete package, you will be prompted to supply information about the HTML server, the application server, the Web server, and the location in which you want to install documentation, the example, and release notes.

You will need to supply the following information:

- Whether to install the software on an application server, a Web server, or both
  - If you choose both, the same software is installed in both places.
Installing Web Deployment Components on UNIX

- The path to your Dynamic 4GL compiler or runtime, as specified in the setting for the FGLDIR environment variable.
  The installation adds binaries to the bin directory, configuration files to the etc directory, and message files to the msg directory under FGLDIR.

- The IP address of the application server.
  The IP address is used to generate the client configuration file, fglcl.conf.

- The type of UNIX system on which the application server is running.
  The prompt displays the system that is assumed. If you select N, it then displays codes for all available system types and allows you to select one.

- The root directory of the Web server.

- The CGI binaries directory of the Web server.

- The IP address of the Web server.

- The type of UNIX system on which the Web server is running.
  The prompt displays the system that is assumed. If you select N, it then displays codes for all available system types and allows you to select one.

- Whether you want to install the documentation (HTML files that describe how to configure and use the Web deployment components).

- The location of the HTML documentation root directory on the Web server.
  The default is /var/httpd/htdocs.
  The installation will not put the documentation in this directory, but will use the directory to propose a new one.

- The path to the directory in which to install the HTML documentation.
  You must specify an absolute path.
  The default is /var/httpd/htdocs/Cli-HTML.

- Whether you want to install the example.

- The path to the directory in which to install the example.
  The default is /usr/fgl2c/cli-html/example.
Installing Web Deployment Components on UNIX

- Whether you want to install the Release Notes
- The path to the directory in which to install the Release Notes
  The default is /usr/fgl2c/cli-html/release.

When you have responded to all the prompts, the installation is complete, and you can proceed to configure your software. Configuration is described in the on-line HTML documentation. For the location, see “Installing the HTML Documentation on the Web Server” on page 1-21.

Manual Installation

Manual installation includes the following tasks:
- Extracting all the files into a temporary directory
- Copying files for the following components to the following locations:
  - The HTML client, to the Web server
  - The HTML server, to the application server
  - The HTML documentation, to the Web server
  - The example, to either server

The rest of this section describes these tasks in more detail.

Extracting the Files

First, create a temporary directory in which to extract the HTML client software, and then extract the files. The binaries and documentation are compressed in the file named ALL/HTML-ALL.TGZ within the CLIENTS/CLI-HTML directory.

If you have the GNU version of the tar program, enter:

```
tar -xzf ./ALL/HTML-ALL.TGZ
```

If you do not have the GNU version of tar, enter:

```
gunzip -c ./ALL/HTML-ALL.TGZ | tar -xzf -
```
After extracting the files, you should see the following directories:

- AppServer
- examples
- release
- WebServer

### Installing the HTML Client on the Web Server

To install the HTML client, you copy two files to the directory where the Web server daemon is running. The client files are initially placed in the `WebServer/cgi-bin/platform-name` directory, where `platform-name` is the specific UNIX or NT platform you are using.

You must copy the following files:

- **fglcl** (the HTML client)
  
  Copy this file to the `cgi-bin` directory under your main Web server directory.

- **fglcl.conf** (the configuration file for the HTML client)
  
  This file contains configuration settings for each Dynamic 4GL application you are running.

  Copy this file to the `cgi-bin` directory under your main Web server directory.

For example, the following code copies each of the files from an installation directory named `/d4gl/Cli-Html` to the CGI binaries directory on a Web server named `/usr3/httpd`, and then sets appropriate file permissions (`SLS-0250` represents the directory where the HTML client binary for the Solaris platform resides):

```bash
cp d4gl/Cli-Html/WebServer/cgi-bin/SLS-0250/fglcl
    usr3/httpd/cgi-bin/fglcl
chmod 755 usr3/httpd/cgi-bin/fglcl

cp d4gl/Cli-Html/WebServer/cgi-bin/SLS-0250/fglcl.conf
    usr3/httpd/cgi-bin/fglcl.conf
chmod 644 usr3/httpd/cgi-bin/fglcl.conf
```
Installing Web Deployment Components on UNIX

Installing the HTML Server on the Application Server

To install the HTML server, you copy four files from the AppServer directory to the directory where your Dynamic 4GL compiler or runtime resides (as specified in the setting for the FGLDIR environment variable).

You must copy the following files:

- **fglhtmlmd** (the HTML server)
  Copy this file to the `bin` directory under `$FGLDIR`.
- **fgl2cres.web** (the resource file for the HTML server)
  Copy this file to the `etc` directory under `$FGLDIR`.
- **fglprofile.web** (the profile for the HTML server)
  Copy this file to the `etc` directory under `$FGLDIR`. You can also specify the FGLPROFILE variable to locate this file.
- **cli-html.iem** (the message file)
  Copy this file to the `msg` directory under `$FGLDIR`.

Optionally, you can also place the **fglcl** and **fglcl.conf** files in the `bin` directory under `$FGLDIR` as a backup for the files on the Web server.

For example, the following code copies each of the three files from an installation directory named `/d4gl/Cli-Html` to the directory on the application server specified by FGLDIR, and then sets appropriate file permissions (SLS-0250 represents the directory where the HTML client binary for the Solaris platform resides):

```bash
    cp /d4gl/Cli-Html/AppServer/bin/SLS-0250/fglhtmlmd
        $FGLDIR/bin
    chmod 755 $FGLDIR/bin/fglhtmlmd

    cp /d4gl/Cli-Html/AppServer/etc/fgl2cres.web
        $FGLDIR/etc
    chmod 644 $FGLDIR/etc/fgl2cres.web

    cp /d4gl/Cli-Html/AppServer/etc/fglprofile.web
        $FGLDIR/etc
    chmod 644 $FGLDIR/etc/fglprofile.web
```
Installing Web Deployment Components on UNIX

Installing the HTML Documentation on the Web Server

The HTML documentation describes the **fglc1.conf** file in more detail, and provides information about using the Web deployment software. To install the documentation, create a subdirectory such as **WebServer/htdocs**, and extract the contents of the **WebServer/doc.tgz** file into this directory. For example:

```
mkdir WebServer/htdocs
cd WebServer/htdocs
tar -xzf ..:/doc.tgz
```

Next, create a directory named **Cli-Html** under the document root directory on your Web server, and copy the documentation there. For example:

```
mkdir /usr3/httpd/htdocs/Cli-Html
cp -r . /usr3/httpd/htdocs/Cli-Html
```

Be sure to name this directory **Cli-Html**; if you use another name, you will need to edit the configuration files for the example program so that the example will run correctly.

You might want to add a link from your home page to the **Cli-Html/index.html** file to make it easy to access the documentation.

Installing the Example

The **phonebook** example is a phone directory that uses the **stores7** database. The example is originally placed in the **example** directory. You can copy it to any directory.

To install the example

1. Place either the NT or the UNIX version of the **phonebook** example in a directory.
2. Run **make** on UNIX, or **nmake** on NT, and follow the on-screen instructions.
3. Enter **make install** (or **nmake install**) to install the data used in the **phonebook** example.
4. Enter `make text` to install the text version of the `phonebook` example, or `make web` to install the Web version.

The text version runs in ASCII and Windows terminals, and can be deployed on the Web; however, it is not optimized for Web deployment.

The Web version includes enhancements for Web deployment.

Configuring your environment to run your applications from the browser involves placing entries in the `fglcl.conf` file and in the `cgi-bin` directory on your Web server. For detailed information on configuring and executing applications, see the on-line HTML documentation. You must make the necessary changes to the configuration files before you can test the installation.

**Troubleshooting the UNIX Installation**

If testing reveals a problem, you can check the HTML client and the HTML server to verify that each is running.

**Checking the HTML Client**

To check the HTML client, you must simulate running your application in a Web server. On the Web server, change to the CGI binaries directory and set the `QUERY_STRING` environment variable to the name you used for your application in the `fglcl.conf` file. For example:

```bash
QUERY_STRING=phonebook
export QUERY_STRING
```

To enable debugging for the client, set the `debug` and `HTMLdebug` parameters in the `fglcl.conf` file as in the following example:

```bash
phonebook.debug=10
phonebook.HTMLdebug=01
```

To run the HTML client, type:

`fglcl`
If you see a display like the following one, the HTML server is not responding (there could also be some HTML code, and the detailed messages can vary from release to release):

```
[DBG-01].**** Debug mode is:10
[DBG-02].****Summarizing configuration from resource file
[DBG-03].****fglserver is 194.150.8.100:98
[DBG-03].****debugstr is 10
[DBG-05].**** -> Sock::init( )
[DBG-05].**** <- Sock::init( )
[DBG-02].**** -> Sock::clientsocket( )
[DBG-02].**** -> Sock::close( ) - closing socket 33
[DBG-02].**** <- Sock::close( ) - status is 0
[DBG-03].**** connect( ) returned a negative value (-1).
[DBG-04].**** Socket Error (null) (-1).
[DBG-02].**** Error in clientsocket:Connection refused(115)
[LOG-04].Error in clientsocket:Connection refused(115)
```

Possible reasons, and possible actions to take, are as follows:

- The HTML server is not running.
  
  Start the server.

- The HTML server is running but is not responding to the client.
  
  Check that the application server name is specified correctly for the `fglserver` parameter in the `fglcl.conf` file.
  
  Check that the application server port is specified correctly in the `appname.conf` file.

- The network is unreachable.
  
  Try to ping the server host from the client. For example:
  
  `ping 158.58.23.30`

  Try to run `telnet` and connect to the server host from the client.
  
  For example:
  
  `telnet 158.58.23.30:1526`

### Checking the HTML Server

To verify that the HTML server is responding to requests, first determine on which port the server is running, by typing the following:

```
network -a
```
The display shows a full listing of your TCP and UDP connections, similar to the following:

tcp 0 0 *:6598 *:* LISTEN

Next, run `telnet` and connect to your application server on the port you have determined it is using:

telnet axis 6598

You should see a display like the following:

Trying 150.55.23.57...
Connected to axis.
Escape character is '^]'.

When you press RETURN, the HTML code for the initial page of the demo application appears to indicate that the server is functioning and communicating with the client:

Pragma: no-cache
Content-type: text/html

<HTML>
META HTTP-EQUIV=REFRESH CONTENT="10;
URL=/cgi-bin/fglccl.exe?demo">
<HEAD>
<BODY BGCOLOR="#F5F5F5">
IMG SRC="Installing Directory Tree"

This section describes the directories created during the installation process.

All of the files installed by the compiler are under the directory specified during the installation and are referenced by the environment variable FGLDIR.
Installation Directory Tree

These directories contain:

- **bin**: The executable files required when you use Dynamic 4GL.
- **bmp**: The pictures included in your 4GL programs running on X11 clients (For the other client interfaces, consult the corresponding section.)
- **clients**: The components that support deploying Dynamic 4GL applications on Windows and the Web.
- **defaults**: Program-specific configuration files.
- **demo**: The Dynamic 4GL demonstration programs.
- **desi**: The configuration manager for X11 clients.
- **etc**: The configuration files and some client resources.
- **etc/ger**: A filter for the German alphabet character set.
- **etc/iso**: A filter for the ISO character set.
- **include**: The `f2c` directory.
- **include/f2c**: The `include` file for C compilation.
- **lib**: The C libraries needed at link time when you create a new runner or compile in C code; also contains the 4GL libraries needed when you compile 4GL programs and modules of some Dynamic 4GL tools.
- **lock**: The data files created by clients running compiled applications. (Removing this directory while applications are running leads to a crash of all the currently running 4GL programs.)
- **msg**: The compiled error and runtime messages handled by the compiler.
Upgrading from Earlier Software Versions

If you previously installed an earlier version of Dynamic 4GL in a directory and now wish to install a newer version in some other directory, unset any environment variables associated with the old version. In particular, unset FGLDIR (or set it to the new directory), PATH, and FGLDBS. You might also need to unset FGLLIBSQL and FGLLIBSYS. Because the installation process should determine the correct value for FGLDBS, FGLLIBSQL, and FGLLIBSYS, it is best to unset them before performing the installation. If you are not using the standard C compiler, cc, for any reason, set the C compiler variables. If you install a new version of Dynamic 4GL over the old directory, it is safest to unset the environment variables.

Microsoft Windows NT Installation

This section describes the prerequisites for installation, the installation process itself, and the configuration tasks. The following is an overview of the NT installation tasks:

1. Installation of Windows NT
2. Installation of Windows NT Service Packs (SP 5 for Windows NT 3.51 and SP 3 for Windows NT 4.0)
3. Installation of a supported Informix server
4. Installation of the Windows front end (formerly the WTK client)
5. Installation of the Dynamic 4GL runtime package for Windows NT
Tip: You can also install components for deploying your applications on the Web. For more information, see “Installing Web Deployment Components on NT” on page 1-53.

Prerequisites for Microsoft Windows NT Systems

Make sure your system meets the following requirements.

C Compiler Prerequisite

The only fully supported C compiler is Microsoft Visual C++ 4.0 or later. A C compiler is only required if you want to call C language functions from 4GL.

Informix Server Prerequisite

As with the UNIX prerequisite, you must have at least one version of the ESQL/C development environment installed to create your own p-code runner. The Dynamic 4GL packages for NT install two default runners. One runner does not include an Informix database interface and is called fglodb. One of three others is also installed, depending on the version of Informix installed on your computer. This runner is named fglrun.

If you want to create your own runner, including calls to external C functions, you will also need a version of the Informix ESQL development package. You must use ESQL/C 7.20.TE1 or higher, because Version 7.20.TD1 may cause system instability on Windows NT 4.0.

TCP/IP Prerequisite

You must install the TCP/IP protocol on the machines that will use Dynamic 4GL. Even if you plan to use the product on a stand-alone machine, TCP/IP features are used.

Important: Only the Microsoft TCP/IP stack is supported. Problems may occur with other TCP/IP stacks.
Step-by-Step Installation on Windows NT

**Hardware Prerequisite**

A network card is required.

*Warning:* Changing the network card disables the license information.

**F4GL Prerequisite**

You are not required to install the Dynamic 4GL WTK client on the machine before installing the Dynamic 4GL runtime package, but it is strongly recommended. The installation software looks for this client and creates icons that allow you to test if the package is installed.

**Step-by-Step Installation on Windows NT**

This section describes how to install Dynamic 4GL on Windows NT.

**Installing the Windows Front End**

For instructions, see “Windows Front-End Configuration” on page 2-8.

**Runtime Installation**

An installation program is provided on the distribution media that automates many of the installation tasks. If you do not have access to this installation program, you will have to install your runtime version manually. For more information about how to install your runner manually, see “Runtime Configuration Section” on page B-3.

To install the runtime package on Windows NT

1. Insert your CD.
   This procedure assumes it is on the D drive.
2. Execute the installation program:
   
   ```dos
   D:\os\nt\runtime\disk1\setup.exe
   ```
3. Continue the installation interactively.
Registering Your License

Dynamic 4GL licenses are made of the following numbers:

- Serial number
- Serial number key
- Installation number
- Installation key

The serial number and serial number key are provided with the Dynamic 4GL media. The installation number is generated for you. You access the installation key by going to the following Web site: www.informix.com/keyissue.

The serial number follows the standard Informix format—an 11-character alphanumeric string. The remaining numbers and keys are always built on the same architecture: 12 uppercase letters and digits followed by an optional checksum number. In order to avoid any confusion, the letter O is never used; it is always the digit 0 (zero).
At the end of the installation process, you will be asked to enter the serial number. The Informix License Manager Program dialog box that appears allows you to install or remove a license, as shown in Figure 1-1.

**Tip:** License information is entered only once. If you need to reinstall the product, you do not need to enter the license information again.

Every field must be completed, with the following exceptions:

- The **Installation Number** field is automatically computed and therefore need not be completed. When you reach that field, a button should appear, allowing you to license using the Web site mentioned at the beginning of this section.

- The **Check** fields are not used and should be ignored.

**Important:** Do not press ENTER to go to the next field. This key validates the **OK** button and would therefore cause the license installation to be incomplete. Use **TAB** or the mouse.
Completing this dialog box will finish the installation process on your Windows NT machine.

**NT Configuration**

Configuration begins after the NT workstation or server is installed and the TCP/IP stack is configured correctly.

**Configuring the Windows NT Machine**

For this example of configuration, the NT server is named ntserver1 (with UNC name `\NTSERVER1`) and has the TCP/IP address 154.120.3.14.

Now create a new user on the NT server. Use the user name magellan, give it the password pacific, and define it as a member of the Users group.
NT Configuration

Figure 1-2
User Manager Window

<table>
<thead>
<tr>
<th>Username</th>
<th>Full Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator</td>
<td></td>
<td>Built-in account for administering the computer/domain</td>
</tr>
<tr>
<td>Guest</td>
<td></td>
<td>Built-in account for guest access to the computer/domain</td>
</tr>
<tr>
<td>informix</td>
<td>informix</td>
<td>informix user account under which OnLine is run</td>
</tr>
<tr>
<td>SOLExecutiveCmdExec</td>
<td>SOLExecutiveCmdExec</td>
<td>SQL Executive CmdExec Task Account</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Groups</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrators</td>
<td>Members can fully administer the computer/domain</td>
</tr>
<tr>
<td>Backup Operators</td>
<td>Members can bypass file security to back up files</td>
</tr>
<tr>
<td>Guests</td>
<td>Users granted guest access to the computer/domain</td>
</tr>
<tr>
<td>SSOLExecutiveAdmin</td>
<td>Database Administrators</td>
</tr>
<tr>
<td>Power Users</td>
<td>Members can share directories and printers</td>
</tr>
<tr>
<td>Replicator</td>
<td>Supports file replication in a domain</td>
</tr>
<tr>
<td>Users</td>
<td>Ordinary users</td>
</tr>
</tbody>
</table>
NT Configuration

Figure 1-3
New User Dialog Box

Figure 1-4
Group Memberships Dialog Box
NT Configuration

You can make a first connection on the NT console as **magellan** to configure the NT environment with this user.

Then, with the **Administrator** account, install an Informix server.

Here is a summary of the important information:

- Machine name—ntserver1
- UNC name—\NTSERVER1
- IP address—154.120.3.14
- Users—**Administrator magellan**

Informix Configuration

For Informix servers, you must set the correct communication protocol configuration and database configuration.

Communication Protocol Configuration

For a Version 7.2x or later Informix server, you must have the following lines in the `services` file located in the directory `%WINDIR%\system32\drivers\etc` on your server and your clients.

On every machine:

```
turbo     1526/tcp
```

On your server:

```
srv_agent  1530/tcp
```

The format is:

```
name       number/tcp
```

For more information on configuring a client-server system, see the **Administrator’s Guide** for your server.
If you use an INFORMIX-CLI connection to the database, the %SYSTEMROOT%\System32\drivers\etc\hosts file needs an entry for each computer on your network. Alternatively, you can configure TCP/IP to use DNS for host name resolution. In this case, you may have to wait for several minutes before the connection to the database is made. For information about these files, refer to your system documentation.

**Database Configuration**

At this point, you have the following components installed: the database server, the ESQL/C package, and INFORMIX-CLI 2.5x.

Restart the system to configure the administrator environment using Informix **Setnet32**. **Setnet32** allows you to set the environment variables and make a connection setup.

On the **Host Information** page, type the following data entries.

![Host Information Page](image)
Click **Apply**.

Click the **Server Information** tab.

Set your machine to be the default server by clicking **Make Default Server**. Then click **Apply**.

On the **Environment** page, check the following variables.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFORMIXDIR</td>
<td>Must be C:\usr\informix</td>
</tr>
<tr>
<td>INFORMIXSERVER</td>
<td>Must be ol_ntserver1</td>
</tr>
<tr>
<td>INFORMIXSQLHOSTS</td>
<td>Must be \NTSERVER1</td>
</tr>
</tbody>
</table>
Click OK.

Now start the database server with the **Administrator** account.
Create a database whose name is **testdbs** and add the connection rights for the user **magellan** using the GRANT statement. Make a connection to it with the **Ilogin** program. When the **Ilogin** demo is running, click the **File** menu. A dialog box is displayed. Set the fields as shown in Figure 1-8.

![Figure 1-8: Login Parameters Dialog Box](image)

### Login Parameters

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>ol_ntserver1</td>
</tr>
<tr>
<td>Hostname</td>
<td>ntserver1</td>
</tr>
<tr>
<td>Servicename</td>
<td>turbo</td>
</tr>
<tr>
<td>Protocolname</td>
<td>olsoctcp</td>
</tr>
<tr>
<td>Username</td>
<td>Administrator</td>
</tr>
<tr>
<td>Password</td>
<td>************</td>
</tr>
<tr>
<td>Stores Database</td>
<td>testdbs</td>
</tr>
</tbody>
</table>

Fill in desired values.

Server, Host, Service, Protocol, User and Password fields will be read from Registry if left blank.

Stores7 will be used if Database field is left blank.

Click **OK**. Another dialog box is displayed with no values. You can click **OK**—there is no problem with the **Administrator** account.

Now, log out from the **Administrator** session and log on with the **magellan** account.

Run the **Setnet32** program.
NT Configuration

On the **Host Information** page, type the following entries.

<table>
<thead>
<tr>
<th>Text Box</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Host</td>
<td>ntserver1</td>
</tr>
<tr>
<td>User Name</td>
<td>magellan</td>
</tr>
<tr>
<td>Password Option</td>
<td>Password</td>
</tr>
<tr>
<td>Password</td>
<td>pacific</td>
</tr>
</tbody>
</table>

Click **Apply**.

On the **Server Information** page, type the following data.

<table>
<thead>
<tr>
<th>Text Box</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informix Server</td>
<td>ol_ntserver1</td>
</tr>
<tr>
<td>Host Name</td>
<td>ntserver1</td>
</tr>
<tr>
<td>Protocol Name</td>
<td>olsoctcp</td>
</tr>
<tr>
<td>Service Name</td>
<td>turbo</td>
</tr>
</tbody>
</table>

Now make a default server by clicking **Make Default Server**. Then click **Apply**.

On the **Environment** page, check the content of the following variables.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFORMIXDIR</td>
<td>Must be C:\usr\informix</td>
</tr>
<tr>
<td>INFORMIXSERVER</td>
<td>Must be ol_ntserver1</td>
</tr>
<tr>
<td>INFORMIXSQLHOSTS</td>
<td>Must be \NTSERVER1</td>
</tr>
</tbody>
</table>

Click **OK**.

Then start the **Ilogin** demo and make a connection to the database **testdbs**.
**NT Configuration**

When the Ilogin demo is running, click the File menu. A dialog box is displayed. Set the different fields as shown in the following table.

<table>
<thead>
<tr>
<th>Text Box</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>ol_ntserver1</td>
</tr>
<tr>
<td>Hostname</td>
<td>ntserver1</td>
</tr>
<tr>
<td>Servicename</td>
<td>turbo</td>
</tr>
<tr>
<td>Protocolname</td>
<td>olsoctcp</td>
</tr>
<tr>
<td>Username</td>
<td>magellan</td>
</tr>
<tr>
<td>Password</td>
<td>pacific</td>
</tr>
<tr>
<td>Stores Database</td>
<td>testdbs</td>
</tr>
</tbody>
</table>

Click OK. Another dialog box is displayed with no values. Click OK—no problem occurs with the magellan account.

**Dynamic 4GL Configuration**

Log on the server with the Administrator account. Run the Graphical Client Setup.
Then run the setup of Dynamic 4GL.

Click Continue.

**Figure 1-9**
Dynamic 4GL Installation Wizard for NT

**Figure 1-10**
Dynamic 4GL Installation Wizard for NT
The installation program will look for the Informix database and ESQL/C version and will localize these products. You can choose between three modes of installation:

- **Automatic search of an existing Informix version.** With this option, the installation searches the database registry for all needed information (database server or INFORMIX-CLI).
- **Specify the Informix directory.** With this option, the installation program prompts for the specified directory for Informix products (database server or INFORMIX-CLI).
- **Informix is not installed.** No Informix product has been installed on your machine (database server or INFORMIX-CLI).

Normally, you will use the automatic search.
At this point, the installation program has found a product. If the path to the Informix product is correct, click **Accept** to continue, or click **Refuse** to go back. If any Informix product is found, the installation program displays a dialog box saying that the installation program will install a runner for the Informix database and a runner for the non-database application.
NT Configuration

By default, the installation program installs the package in the %windir% directory (on the disk where the system is installed). If you want to change the directory where Dynamic 4GL will be installed, click **Browse**.

After you click **Browse**, this dialog box is displayed. Select the directory where you want the product to be installed and click **OK**.

Click **Next** to start the installation procedure.
At the end of the installation procedure, if no license is installed you will be prompted to register your license. During the licensing procedure, do not press ENTER or RETURN. You must use TAB to go from one field to another.

After licensing, if you have installed the development package, the program compiles the different p-code libraries needed to compile 4GL to p-code. For the runtime license, this compilation is not needed because you will never have to compile a program.

Now that Dynamic 4GL is installed, you can configure the product.

From the Start menu, choose Programs➞Dynamic 4GL➞Dynamic 4GL Workshop. The command prompt window appears, with all parameters configured for the Administrator account.

You can now make the environment file for the user magellan.

Copy the %FGLDIR%\env.bat file to %FGLDIR%\magellan.bat.

In the magellan.bat file, change the following lines:

```
SET FGLPROFILE=C:\usr\FGl2C\ETC\FGLPROFILE
SET LOGNAME=Administrator
```

to

```
SET FGLPROFILE=C:\usr\FGl2C\ETC\magellan.prf
SET LOGNAME=magellan
```

Save the magellan.bat file.

Now copy the %FGLDIR%\etc\fglprofile file to %FGLDIR%\etc\magellan.prf.

Make the sample program testdbs.4gl to test the database connection; for example:

```
MAIN
    Database testdbs
    Display "Status: ", status
END MAIN
```

Compile this program by using the following command:

```
C:\usr\fgl2c fgl2p -o testdbs.42r testdbs.4gl
```
NT Configuration

Now run the program with this command:

```
C:\usr\fgl2c fglrun testdbs.42r
```

If the database server is started, you will see the following message:

```
Status:0
```

This means that the connection to the database testdbs is running correctly.

Now log out from the Administrator account and log on with the magellan account. Go into the %FGLDIR% directory. Run the magellan.bat program to set the environment variables. Run the program testdbs.42r. You must have the same display as before if the database server is started.

Here is summary of the important information:

<table>
<thead>
<tr>
<th>FGLDIR</th>
<th>C:\usr\fgl2c</th>
</tr>
</thead>
<tbody>
<tr>
<td>FGLPROFILE</td>
<td>%FGLDIR%\etc\fglprofile for the Administrator user</td>
</tr>
<tr>
<td></td>
<td>%FGLDIR%\etc\magellan.prf for the magellan user</td>
</tr>
<tr>
<td>LOGNAME</td>
<td>Administrator for the Administrator user</td>
</tr>
<tr>
<td></td>
<td>magellan for the magellan user</td>
</tr>
<tr>
<td>FGLSERVER</td>
<td>154.120.3.14:0 for both users</td>
</tr>
<tr>
<td>Environment</td>
<td>env.bat for the Administrator user</td>
</tr>
<tr>
<td>files</td>
<td>magellan.bat for the magellan user</td>
</tr>
</tbody>
</table>

Ataman Configuration

To unzip the Ataman package in the C:\usr\ataman directory, execute the following command:

```
C:\usr\ataman atrlis install
```

Next, click the Ataman icon in the Control Panel.
### Ataman TCP Remote Logon Services

**Ataman® TCP Remote Logon Services**

All rights reserved.

**FOR LICENSING INFORMATION, SEE THE FILE LICENSE.TXT.**

**Version 2.4**

- Contact Information
  
  Ataman Software, Inc.
  
  http://www.ataman.com
  
  support@ataman.com
  
  (970)-225-3131

*Ataman® is a trademark of Ataman Software, Inc.*

---

**Figure 1-15**
Ataman TCP Remote Logon Services Dialog Box
To configure the user connection, click the **Users** folder and then click **Add User**.

*Figure 1-16*  
Add User Dialog Box
Installing Dynamic 4GL

Figure 1-17
Banners Page

NT Configuration
NT Configuration

Click the **Advanced** tab.

![Ataman TCP Remote Logon Services](image)

Remove the asterisk (*) in the **List of hosts allowed to connect** text box and click **OK**.

Now you can run the Ataman remote connection by running the following program at the command prompt:

```
C:\usr\ataman atrls start
```
Before testing the connection, you must check if the following variable is set in your `fglprofile`:

```
fglrun.database.listvar = CC8BITLEVEL COLLCHAR CONRETRY CONTIME DBANSIWARN DBDATE DBLANG DBMONEY DBNLS DBPATH DBTEMP DBTIME DELIMIDENT ESOLMF FET_BUFF_SIZE GL_DATE GL_DATETIME INFORMIXDIR INFORMIXSERVER INFORMIXSQLHOSTS LANG LC_COLLATE LC_CTYPE LC_MONETARY LC_NUMERIC LC_TIME DBALSBC DBAPICODE DBASCIIBC DBCENTURY DBCODESET DBCCONNECT DBCSCONV DBCSOVERRIDE DBCSWIDTH DBFLTMSK DBMONEYSCALE DBSS2 DBSS3
```

You must also change some values in the configuration file `magellan.prf`:

```
fglrun.remote.envvar = REMOTEADDRESS
```

The first two lines of the following code must be uncommented, and the next two lines must be added:

```
fglrun.setenv.0 = INFORMIXSERVER=ol_ntserver1
fglrun.setenv.1 = INFORMIXHOST=ntserver1

fglrun.defaultenv.0 = INFORMIXDIR=C:\usr\Informix
fglrun.defaultenv.1 = INFORMIXSQLHOSTS=\\NTSERVER1
```
Save these modifications. Then create a connection using **Wtk**.

**Figure 1-19**
Establish Connection Dialog Box

**Figure 1-20**
Installing Web Deployment Components on NT

When the connection is made, enter the password. You will be in the \%FGLDIR\% directory. Start your environment file magellan.bat. Now you can run testdbs.42r. This program will indicate that your status is set to 0, meaning that the connection was successful.

This completes the configuration of your NT server.

Installing Web Deployment Components on NT

Installation on NT systems involves running an InstallShield wizard. This section provides information to help you respond to the prompts.

Prerequisites for Installing the Web Deployment Components

To install this software, you need the following:

- A running Dynamic 4GL compiler with a valid license number
- A running Web (HTTP) server
- A browser with HTTP form and table support (HTTP version 3.2 or higher)

These features are present in Microsoft Internet Explorer, Version 2.x or higher, and in Netscape Navigator, Version 2.x or higher.

Once you have completed installing the Dynamic 4GL software, the components for Web deployment are available in the /CLIENTS/CLI-HTML directory on the CD. The following subdirectories are present:

- ALL
- BIN
- DOC
- SELFEXTR
- TLB

In addition, you will find the cli-html.exe file, which is the executable for the installation program.

Running the Installation Program

To run the program that extracts the files for the HTML client software on NT, execute the file named cli-html.exe. The program extracts the files into a temporary directory you specify.
Installing Web Deployment Components on NT

At each screen, respond to the prompts and click Next to continue extracting the files.

When prompted, you will need to supply the following information:

- A location for the HTML client (Choose Destination Location screen)
  The default is C:\4glsrv\Cli-HTML.
  You can change the installation directory from the default, but make sure you do not specify the same directory for the compiler. (Be sure that the location specified by the %FGLDIR% environment variable is not the location you give for Choose Destination Location.)

- The type of installation (Setup Type screen):
  - If you select Complete installation, all the components are installed: HTML client, HTML server, documentation, and the example.
  - If you select Customized installation, you will be prompted for the package to install in the Select Components screen. To specify a package, check the check box next to it. The available packages are described immediately after this list.

- The program folder in which the startup icon resides
  By default, the startup icon is created in the Programs section of the Start menu.

You must then manually copy the various components from the temporary directory to the appropriate locations on the Web server and application server. The procedures for copying the files are the same as those described for UNIX systems, beginning with "Installing the HTML Client on the Web Server" on page 1-19.

Available Packages

Customized installation lets you select any of the following packages:

- Shared libraries. These libraries are required for executing the software under Windows NT.
  If you already have these libraries on your system, the installation replaces them only if the libraries in the installation have more recent versions than the existing ones.
Installing Web Deployment Components on NT

- **HTML documentation.** These files provide configuration and usage information for deploying applications on the Web.
- **Client and server for Windows NT.** These include HTML client and HTML server only.
- **Client and server for AIX, HP-UX, IRIX, SCO, Sun Solaris (Sparc), Unixware, Linux.** This package is for UNIX systems that you can download and configure manually. Download one of these options for installation on a remote application server or Web server.
- **Example.** For more information about the on-line example, see “Installing the Example” on page 1-21. The information applies equally to UNIX and NT.

**Troubleshooting the NT Installation**

If testing reveals a problem, you can check the HTML client and the HTML server to verify that each is running. For complete information, first see “Troubleshooting the UNIX Installation” on page 1-22. This section gives differences that apply to NT.

**Checking the HTML Client**

To set the QUERY_STRING environment variable on NT for the phonebook example, type:

```
set QUERY_STRING=phonebook
```

**Checking the HTML Server**

To see the TCP and UDT connection listing, type:

```
C:\> netstat -a
```

**Configuring Your System**

Configuring your environment to run your applications from the browser involves placing entries in the fglcl.conf file. This file is located in the cgi-bin directory of the Web server. For detailed information on configuration, see the on-line HTML documentation.
Configuring Your Client Environment

In This Chapter ........................................ 2-3
  UNIX Front-End Configuration .......................... 2-3
    Installing the X11 Front End ....................... 2-4
      Installing TCL/TK ................................ 2-4
      Installing the Daemon ............................. 2-5
    Managing Application Windowing ................. 2-5
  Windows Front-End Configuration .................... 2-8
    Installing the Windows Front End ................. 2-8
    How to Use the Windows Front End ............... 2-9
In This Chapter

The front end is a small application that manages the interface between the client and the server. This chapter describes the installation, configuration, and use of the front end.

Two front ends are available: the X11 front end, for all UNIX graphical interfaces, and the Microsoft Windows front end, for Windows 3.11, Windows 95, and Windows NT 3.51 or later.

UNIX Front-End Configuration

The X11 front end allows you to run your 4GL programs in graphical mode with all the X11-compliant interfaces. This front end is composed of a TCL/TK interpreter and a daemon that manages communication between the TCL/TK interpreter and the 4GL runner. Communication between the 4GL program and the daemon and between the daemon and the X11 interface uses the TCP/IP protocol. Therefore, these components can be installed on different machines.

Your X11 interface should be configured to display at least 256 colors for the best output quality.
Installing the X11 Front End

The TCL/TK interpreter is included on the Dynamic 4GL distribution CD. The daemon called fglX11d is included in both the UNIX Dynamic 4GL development package and the runtime package.

Installing TCL/TK

If you have a CD-ROM drive connected to your UNIX machine, mount the CD on your file system:

```bash
$ mount your_cdrom_device_name /cdrom
```

**Tip:** Depending on your system, the syntax of the mount command may be different. Check your UNIX manual. Also, note that depending on your system, the name of the files located on the CD may be in either lowercase or uppercase letters.

Then go to the `/cdrom/OS/UNIX` directory and enter the following command to start the installation process:

```bash
$ sh ./INSTALLTCL.SH -i
```

The `-i` option starts the installation of TCL/TK. If you do not specify this flag you will get the syntax help message.

The installation shell first tries to determine your operating system name and checks for a few requirements. You do not need to run this shell as root. You only need to have sufficient permissions to create the directories where you want to install the TCL/TK package. This shell also checks if you already have a TCL/TK package installed.

It then prompts you for the installation directory and starts copying the files from the CD to your hard drive.

After the files are installed, the installation process prompts you for a directory where the shell script `envtcl` is to be created. This script will be used later to set all the needed environment variables to make the TCL/TK interpreter work. This script is written in Bourne shell. Changing it to C shell requires some adaptive work.

The two variables set by the `envtcl` file are:

- **TCL_LIBRARY**—the path to the `tcl` libraries
- **TK_LIBRARY**—the path to the `tk` libraries
It also adds the /bin subdirectory to the PATH variable.

If you do not have a CD-ROM drive on your UNIX system, simply copy the file tcltk.sh from the directory /OS/UNIX/your_OS_name/SELFEXTR to your UNIX system. Note that you must use binary transfer (8-bit) and not ASCII transfer (7-bit) mode.

To start the installation, execute the following command:

```
$ sh ./TCLTK.SH -i
```

The -i option starts the installation of TCL/TK. If you do not specify this flag you will get the syntax help message.

The needed files for the installation are extracted from the shell script, and the same process occurs as was previously described.

After the installation is complete, execute the envtcl shell script to set the correct environment to use the TCL/TK interpreter. Add a call to this script in one of your startup files (.profile or .login).

**Installing the Daemon**

The daemon fglX11d is always installed with the Dynamic 4GL development or runtime package. This daemon is located in the $FGLDIR/bin directory.

**Managing Application Windowing**

The fglX11d daemon process manages application windowing. One occurrence of this daemon must be started for each different X11 display. This daemon can run on a machine other than the one where the 4GL program runs or the one where the output is displayed.
Managing Application Windowing

The fglX11d daemon uses a TCP/IP socket to communicate with the 4GL program and uses the X11 standard DISPLAY mechanism to specify the output interface.

The 4GL program opens a TCP/IP socket to the address and port number specified by the FGLSERVER environment variable. This variable must be set in the environment where the 4GL program will be started. The syntax of the FGLSERVER variable is:

```
machine_ip_address:daemon_number
```

where `machine_ip_address` is set to the TCP/IP address or name of the machine running the fglX11d daemon, and `daemon_number` is set to the occurrence number of the fglX11d daemon. Each daemon started on one machine should have a unique occurrence number. Here is an example in Bourne shell:

```
$ FGLSERVER=127.0.0.4:9
$ export FGLSERVER
```

These two lines tell the compiler that the graphical daemon will run on the machine with the IP address 127.0.0.4 and use the daemon number 9.
The second needed variable is the **DISPLAY** variable, which tells the **fglX11d** daemon which X11 server it must use for the graphical output (to your client machine). This variable must be set in the environment where the **fglX11d** daemon will be started. The syntax of the **DISPLAY** variable is:

```
machine_ip_address:X_server
```

where **machine_ip_address** is set to the TCP/IP address or name of the client machine, and **X_server** is set to the number of the X server that the client will use. If you want to use the X11 server running on the machine with the IP address 127.0.0.5 with the X11 server number 0, use the following commands:

```
$ DISPLAY=127.0.0.5
$ export DISPLAY
```

The last step is to start the **fglX11d** daemon. Here is the syntax:

```
$ fglX11d [-n daemonNumber] [-w wishName] [-f scriptName] [-s portNumber] [-e daemonNumber] [-l] [-v] [-a]
```

The following table describes the options in this command.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-n daemonNumber</td>
<td>Single ID to identify multiple daemon occurrences on one host (default: 0)</td>
</tr>
<tr>
<td>-w wishName</td>
<td>Name of the visual shell to be used (default: wish)</td>
</tr>
<tr>
<td>-f scriptName</td>
<td>Name of the script to be used to initialize the server (default: $FGLDIR/etc/fgl2c.tcl)</td>
</tr>
<tr>
<td>-s portNumber</td>
<td>TCP port to be used (default: 6400)</td>
</tr>
<tr>
<td>-e daemonNumber</td>
<td>Occurrence number of the daemon to shut down (default: 0)</td>
</tr>
<tr>
<td>-l</td>
<td>Logs all traffic to stderr</td>
</tr>
<tr>
<td>-v</td>
<td>Gives the version information and exits</td>
</tr>
<tr>
<td>-a</td>
<td>Gives the number of the next free daemon</td>
</tr>
</tbody>
</table>

Since this is a daemon, you should run the process in the background by adding the & symbol at the end of the command line. To stop a started daemon, use the -e option of the **fglX11d** daemon.
Windows Front-End Configuration

The Windows front end, also often called the WTK client, allows you to run 4GL programs in graphical mode with all Windows systems. In contrast to the X11 front end, the communication daemon and the TCL/TK interpreter are included in the WTK client. The Windows machine should have at least 256 colors and must be able to run 32-bit programs (386 or better).

Installing the Windows Front End

The client machine where WTK is installed must have a Microsoft TCP/IP stack installed and must be able to run 32-bit applications. Therefore, if you are running Windows 3.11 you need to install the WIN32S extension.

On the Dynamic 4GL CD, you will find the WIN32S extension and a TCP/IP stack for Windows 3.11 machines. To install the WIN32S extension, go to the directory \WINDOWS\UTIL\WIN32S\DISK1 of the CD, execute the setup.exe program, and follow the instructions on the screen.

To install the TCP/IP 32 stack from Microsoft on Windows 3.11 machines, go to the directory \WINDOWS\UTIL\TCP/IP32\TCP/IP32 and read the instructions given by Microsoft in the readme.txt file.

Once your client machine is ready for the installation of the Windows front end, close all applications.

To install the WTK client from the Dynamic 4GL CD, run the program \WINDOWS\WTK\DISKversion_number\setup.exe. Follow the screen instructions to complete the installation of the WTK client.
How to Use the Windows Front End

The Windows front end includes the communication daemon and the TCL/TK interpreter. Therefore, you do not need to install any front-end software on your server.

Start a telnet or an rlogin connection from the Windows machine to the server machine where Dynamic 4GL is installed. Although you could use the WTK 4GL-Server with TCP/IP terminal emulation, the WTK-Rlogin terminal emulation, which is part of the delivery, has several advantages:

- Automatic startup of 4GL-Server when starting a terminal
- Processing of commands on the server after starting the terminal
- 3D presentation of the character-based application on the server

To create an icon for a terminal connection, indicate which icon to use, and supply the name of the remote user and terminal type. Optionally, you can enter a command string that contains commands for the remote computer, separated by semicolons. This works only when opening a connection that does not require a password.

WTK-Rlogin does not use the Telnet-Internet Service; instead, it uses the Login-Internet Service, which uses the trusted host principle (there are machines that the rlogin daemon trusts) and authorizes you to build a connection without password transmission.

On UNIX machines, this service is used with the rlogin program.

To connect using WTK-Rlogin, there must be an Rlogin-Internet daemon running on the UNIX machine (there must be a line login/tcp in file /etc/services, and a line in file /etc/inetd.conf must start with login).

For more information on creating connections, see the wtksrv1.wri file in the installation directory of the WTK client.
How to Use the Windows Front End

Now that you are connected to the server, specify the name or address of the client machine and the occurrence number of the communication daemon running on the Windows machine. As with the X11 front end, you use the FGLSERVER environment variable. The syntax of the FGLSERVER variable is:

```
machine_ip_address:daemon_number
```

where `machine_ip_address` is set to the TCP/IP address or the name of the machine running the Windows front end, and `daemon_number` is set to the occurrence number of the front end. Here is an example in Bourne shell:

```
$ FGLSERVER=127.0.0.10:0
$ export FGLSERVER
```

These two lines tell the compiler that the graphical daemon will run on the machine with the IP address 127.0.0.10 and use the daemon number 0. See the `install_dir\wtsrv1.wri` file for instructions on specifying the occurrence number of the Windows front end.
Using Dynamic 4GL

Chapter 3  Overview of Dynamic 4GL
Chapter 4  Using the Compiler
Chapter 5  Differences Between 4GL and Dynamic 4GL Compilers
Chapter 6  Extensions to the 4GL Language
Overview of Dynamic 4GL

In This Chapter .............................................. 3-3
About Dynamic 4GL ........................................ 3-3
Basic Principles ............................................. 3-4
   Software Architecture .................................. 3-4
   P-Code and C Code ..................................... 3-4
Compilation Tools .......................................... 3-5
   Main Compilation Tools ................................. 3-5
   Other Compilation Tools ............................... 3-6
   Configuration Tools ................................... 3-6
   Miscellaneous Programs and Scripts ............... 3-7
Required Environment Variables ......................... 3-7
Compiling a Simple Program ................................ 3-8
   Setting Up the Environment ......................... 3-9
   Writing the Source Code .............................. 3-10
   Compiling the Source Code ......................... 3-11
      Compiling to P-Code ............................... 3-11
      Compiling to C Code .............................. 3-12
      Compiling the Form File ......................... 3-12
   Running the Program on the Front End ............... 3-13
      Running the Example on the X11 Front End ....... 3-13
      Running the Example on the Windows Front End .. 3-16
In This Chapter

This chapter describes the principles of Dynamic 4GL and gives a short description of the main tools you will use while developing applications.

About Dynamic 4GL

Dynamic 4GL is an innovative product, developed using object-oriented technology. Its advanced design opens new possibilities for companies with existing INFORMIX-4GL applications.

By recompiling your 4GL source code with Dynamic 4GL, you can transform your existing text-based application into a thin client/server system.

Furthermore, you can generate pseudo-executable code (p-code) on most UNIX machines and on Windows NT. The client machines can be Windows (3.11, 95, or NT) or UNIX (X window system).

You can execute your applications in graphical or ASCII mode by changing a single environment variable at runtime. This allows you to control the migration rate of the client machines.

The Dynamic 4GL compiler has many advantages over all other new development software and languages. The other tools require extensive retraining, long development cycles, and multiple source code requirements. But the biggest advantage of using Dynamic 4GL is that you can keep using the applications you have developed and perfected over the years with INFORMIX-4GL—the applications that have the best functionality and behaviors for your customers. So keep your well-designed INFORMIX-4GL routines and add the latest development features with the extensions of Dynamic 4GL.
Basic Principles

This section describes the basic architecture of 4GL application environments, and it points out the differences between p-code and C code.

Software Architecture

The standard architecture of 4GL application installations is to have Dynamic 4GL installed on the application server, in its development or runtime version, along with the 4GL programs. The application server can be a UNIX server or a Windows NT machine. Most of the time, the database server is installed on this same machine, but this is not required. The database interface is only required on the database server and the application server (the machine that will run the 4GL program).

The client machines can be X11 compliant interfaces or Windows machines. Each kind of client has its own Dynamic 4GL daemon that handles all the graphical interface aspects of the 4GL applications.

P-Code and C Code

The 4GL software sources can be compiled to either p-code or C code. P-code is preferable. It is not possible to run C code programs on Windows machines. C code is the result of the transformation of your 4GL source code into C and then from C into a hardware-dependent executable code, whereas p-code is hardware-independent pseudo-executable code. The same p-code can be executed on any of the operating systems on which Dynamic 4GL is installed. Furthermore, p-code allows you to use many of the improvements added to 4GL that are not available for use with C code.
Compilation Tools

All the tools you need in order to compile 4GL programs to p-code or C code are located in the `bin` subdirectory. These tools are described in the following sections.

Main Compilation Tools

The following table lists the programs you will most often use to compile applications.

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fgl2p</td>
<td>Script to compile applications to p-code</td>
</tr>
<tr>
<td>fgl2c</td>
<td>Script to compile applications to C code</td>
</tr>
<tr>
<td>fglcomp</td>
<td>Main compiler program</td>
</tr>
<tr>
<td>fgllink</td>
<td>Main linking program</td>
</tr>
<tr>
<td>fgllform</td>
<td>Tool for compiling form specification files (.per)</td>
</tr>
<tr>
<td>fglschema</td>
<td>Script to create a schema of your databases used by the 4GL compiler at compile time</td>
</tr>
<tr>
<td>fgllmkrun</td>
<td>Script to create a new p-code runner</td>
</tr>
<tr>
<td>fgllnodb</td>
<td>The default p-code runner without any database interfaces</td>
</tr>
<tr>
<td>fgllrun</td>
<td>The p-code runner created during the installation process, including your Informix interface</td>
</tr>
</tbody>
</table>
Other Compilation Tools

Use the following scripts to create archives and locate libraries.

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar42o</td>
<td>Script to create archive files from .42o object files</td>
</tr>
<tr>
<td>findlib.sh</td>
<td>Script to find all the libraries needed on your system to create p-code runners or C code executables</td>
</tr>
</tbody>
</table>

Configuration Tools

The following table lists tools that aid in configuration.

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fglmkmsg</td>
<td>Tool to create the runtime error message libraries</td>
</tr>
<tr>
<td>licencef4gl</td>
<td>Script to install a license</td>
</tr>
<tr>
<td>confdesi</td>
<td>Script to start the configuration program for the X11 interface</td>
</tr>
<tr>
<td>fglfontsel</td>
<td>Font selection tool for X11 interfaces (p-code version)</td>
</tr>
<tr>
<td>fglfontsel.42e</td>
<td>Font selection tool for X11 interfaces (C code version)</td>
</tr>
</tbody>
</table>
Miscellaneous Programs and Scripts

This table lists other helpful tools.

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtsinstall</td>
<td>Script to create the p-code libraries and to compile the Dynamic 4GL tools to p-code</td>
</tr>
<tr>
<td>fglinstall</td>
<td>Script to create the C code libraries and to compile the Dynamic 4GL tools to C code</td>
</tr>
<tr>
<td>fglpager</td>
<td>Script to start the graphical editor used to display reports (p-code version)</td>
</tr>
<tr>
<td>pager.42e</td>
<td>Graphical editor used to display reports (C code version)</td>
</tr>
<tr>
<td>install.sh</td>
<td>Script used during the installation of packages and patches</td>
</tr>
<tr>
<td>fglX11d</td>
<td>Graphical daemon for the X11 interfaces</td>
</tr>
<tr>
<td>fglWrt</td>
<td>Main license program</td>
</tr>
</tbody>
</table>

Required Environment Variables

This section describes the most important variables needed to run the Dynamic 4GL compiler and your 4GL programs. During UNIX installations, a file is created that sets the environment for your configuration. This file is located in the directory where you have installed Dynamic 4GL and is named envf4gl. This file is a Bourne shell script. If you are using C shell you must configure the file to work on your system. It is a good idea to include it in your user configuration files.

The FGLDIR environment variable is mandatory for all calls to all programs and scripts of the Dynamic 4GL compiler and for making your Dynamic 4GL programs run. It should be set to the directory where the Dynamic 4GL compiler is installed.

Tip: For the rest of this manual, the directory where Dynamic 4GL is installed will be called SFGLDIR, in reference to the UNIX naming convention.
Compiling a Simple Program

The following table lists the other environment variables you need to set.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FGLGUI</td>
<td>This variable is your switch for executing your programs in graphical mode or in ASCII mode. The choice is made at runtime and does not need to be made at compile time.</td>
</tr>
<tr>
<td>PATH</td>
<td>For convenience, add the /bin subdirectory where the compiler is installed.</td>
</tr>
<tr>
<td>FGLSERVER</td>
<td>This variable is used at runtime to specify the location of the graphical interface server. This variable is made of the IP address and the number of the TCP/IP socket port to use for communication between the server and the client. These numbers are separated with a colon.</td>
</tr>
</tbody>
</table>

With the Bourne shell, use the following syntax to set the variables:

```bash
$ FGLDIR=/usr/fgl2c
$ export FGLDIR
```

With C shell, use the syntax:

```bash
$ setenv FGLDIR /usr/fgl2c
```

Compiling a Simple Program

This section provides a step-by-step procedure to compile a simple 4GL program with the Dynamic 4GL compiler, which involves these tasks:

- Running the `env4gl` script to set up the environment
- Writing the 4GL source code
- Compiling to p-code or C code

After you compile the program, you compile the form specification file.
Setting Up the Environment

The first task is to set up a correct environment configuration for compilation. If you used the automatic installation, log on to your UNIX machine with a user account that has permission to use the compiler. Then execute the `envf4gl` script located in the `$FGLDIR` directory (assuming that the compiler is installed in the `/usr/fgl2c` directory):

```
$ cd /usr/fgl2c ($ . ./envf4gl
```

This script adds the following environment variables to your environment.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FGLDIR</td>
<td>The directory specified during the installation for Dynamic 4GL</td>
</tr>
<tr>
<td>INFORMIXDIR</td>
<td>The Informix root directory specified during the installation of Dynamic 4GL</td>
</tr>
<tr>
<td>FGLDBS</td>
<td>Setting this tells to the compiler which version of the Informix interface you have installed on your computer</td>
</tr>
<tr>
<td>FGLCC</td>
<td>Name of the C compiler you want to use</td>
</tr>
<tr>
<td>FGLLIBSQL</td>
<td>List of the needed Informix SQL libraries</td>
</tr>
<tr>
<td>FGLLIBSYS</td>
<td>List of the needed system libraries</td>
</tr>
<tr>
<td>FGLGUI</td>
<td>Used only at runtime to specify if the program should be executed in graphical mode or in ASCII mode</td>
</tr>
<tr>
<td>PATH</td>
<td>The <code>$FGLDIR/bin</code> directory is added to your PATH</td>
</tr>
</tbody>
</table>
Writing the Source Code

The second step is to write the 4GL source code for your application. Here is a sample program made of two 4GL modules and one form specification file:

The first source code file, **ex1-1.4gl**:

```plaintext
MAIN
CALL fgl_init4js()
OPEN WINDOW w1 AT 1,1 WITH 24 ROWS, 80 COLUMNS
OPEN FORM frm1 FROM "ex1-1"
DISPLAY FORM frm1
MENU "F4GL"
   COMMAND "Message box"
      CALL message_box()
   COMMAND "Exit"
      EXIT MENU
END MENU
END MAIN
```

The second 4GL source code file, **ex1-2.4gl**:

```plaintext
FUNCTION message_box()
DEFINE f01,f02,bt1 CHAR(20)
INPUT BY NAME bt1,f01,f02;
CALL fgl_winmessage(f01,f02,bt1)
END FUNCTION
```

The form specification file, **ex1-1.per**:

```plaintext
DATABASE FORMONLY
SCREEN {
    Icon [bt1 ] Title[f01]
    Message[f02]
}

ATTRIBUTES
f01 = formonly.f01;
f02 = formonly.f02;
bt1 = formonly.bt1.widget="RADIO", default="info",
    config="info Info exclamation Exclamation question
    Question stop Stop";
```

Note that all the strings between double quotes are case sensitive.
Compiling the Source Code

The last step is to compile this 4GL source code. You can compile to either p-code or C code.

Compiling to P-Code

The advantages of p-code over C code are numerous. One of the main advantages is that p-code is portable: you compile it once and then you can run the same compiled modules on every machine on which a Dynamic 4GL runtime package is installed. Furthermore, all the new 4GL features are implemented in p-code only.

Here is how to compile ex1-1 to p-code. Change to the directory where you have created the ex1-1.4gl and ex1-1.per files. Check that your environment is correctly set:

```
$ echo $FGLDIR
```

This statement should return the directory where you have installed Dynamic 4GL. Then also check if the $FGLDIR/bin directory is included in the PATH variable:

```
$ echo $PATH
```

Now compile the .4gl source code files into modules with the .42m extension. Use the fgl2p script calling the fglcomp program:

```
$ fgl2p ex1-1.4gl
$ fgl2p ex1-2.4gl
```

After compiling, you must link the two .42m modules together into a file with the .42r extension. Use the fgl2p script again, but this time it calls the fgllink program:

```
$ fgl2p -o ex1.42r ex1-1.42m ex1-2.42m
```

The resulting ex1.42r file does not contain any executable code. This file is a hash table containing calls to the functions included in the .42m modules. This is why it is absolutely necessary to keep these modules accessible at runtime.
Compiling the Form File

**Compiling to C Code**

You can also compile this program to C code if you choose. C code is only available for UNIX platforms. In fact, the command lines are exactly the same as for p-code compilation. The only difference is that you use `fgl2c` instead of `fgl2p`. The main difference is the need for a C compiler, a linker, and two extra environment variables: `FGLLIBSYS` and `FGLLIBSQL`. These two variables are defined at install time in the `envf4gl` file located in the `$FGLDIR` directory. Check that they are correctly set:

```
$ echo $FGLLIBSYS
$ echo $FGLLIBSQL
```

These two commands should return a list of system libraries and Informix libraries.

To compile the `.4gl` source code files into object files with the `.42o` extension, the `.4gl` files are first compiled into `.42c` files by the `fglcomp` program and then are compiled by your C compiler into `.42o` object files:

```
$ fgl2c -c ex1-1.4gl
$ fgl2c -c ex1-2.4gl
```

Note that in this case, you should use the `-c` flag.

Next, link the object files, your system libraries, the Dynamic 4GL libraries, and the Informix libraries together into a single executable file with the `.42e` extension:

```
$ fgl2c -o ex1.42e ex1-1.42o ex1-2.42o
```

**Compiling the Form File**

Form files are compiled with the `fglform` compiler. The compiled forms are used by p-code programs as well as by C code programs. To compile the form specification file `ex1-1.per`, type the following command:

```
$ fglform ex1-1.per
```

The result of the compilation is a `.42f` file. In this case, you get the file `ex1-1.42f`.  

---

3-12 Dynamic 4GL User Guide
Running the Program on the Front End

This section describes how to configure the front end, whether on UNIX or Windows, to run the sample program you just compiled.

Running the Example on the X11 Front End

Now you can use the X11 front end to execute the program you just compiled. For this example, assume that the compiler and the 4GL programs are on a UNIX server with a TCP/IP name set to water, and assume that the program is displayed on the X11 interface of a machine named fire. The client, fire, has a telnet or rlogin connection to the water machine.

You must now add the TCL/TK environment variables with the envtcl shell script. Suppose that the TCL/TK is installed in the /usr/local/tcltk directory and that the envtcl file is also located in this directory:

```bash
$ cd /usr/local/tcltk
$ . ./envtcl
```

You can check that the environment is correctly configured with the following two commands:

```bash
$ echo $TCL_LIBRARY
$ echo $TK_LIBRARY
```

Now set the DISPLAY environment variable so that water knows to send all graphical output to the machine fire.

```bash
$ DISPLAY = "fire:0"
$ export DISPLAY
```

All graphical programs started in this environment will now be displayed by the X11 server number 0 of the machine fire. You can start the TCL/TK interpreter to check if it is correctly installed and if the DISPLAY variable is correctly set:

```bash
$ wish
```

Your shell prompt should turn into a percent character (%), and a small black square should appear. You are now in the TCL/TK Interpreter. To quit, enter:

```bash
% exit
```
Running the Program on the Front End

The small square should disappear, and the prompt should be restored to your standard UNIX prompt.

Because you are the first user to run graphical 4GL programs on the machine water, you could choose the number 0 as the occurrence number for the fglX11d daemon. But for this example, use 5 as the occurrence number:

$ fglX11d -n 5 &

Adding the ampersand character (&) to the command line causes the daemon to be executed in the background so that the prompt can be available for new commands.

You can now add the FGLSERVER variable to tell the 4GL program which daemon to use. In this case, the daemon is running on the machine named water and the occurrence number is 5:

$ FGLSERVER=water:5
$ export FGLSERVER

The last thing to check is the value of the FGLGUI variable. If it is set to 0 the program will be executed in ASCII mode exactly as if compiled with INFORMIX-4GL compilers. If set to 1 it will use the fglX11d daemon, and the application will be displayed in graphical mode. Check the setting with the following command:

$ echo $FGLGUI
Running the Program on the Front End

If it is set to 0 or if it is not set, set it to 1 with the following commands:

$ FGLGUI=1
$ export FGLGUI

Now that the environment is correctly set, you can start the application. Change to the directory where the program is compiled. If you have compiled it to p-code, the following files are in the directory.

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>The source files</td>
<td>ex1-1.4gl, ex1-2.4gl, ex1-1.per</td>
</tr>
<tr>
<td>The compiled form</td>
<td>ex1-1.42f</td>
</tr>
<tr>
<td>The p-code modules</td>
<td>ex1-1.42m, ex1-2.42m</td>
</tr>
<tr>
<td>The p-code link</td>
<td>ex1.42r</td>
</tr>
</tbody>
</table>

To start the program, type the name of your runner and, as the first parameter, the name of the file resulting from the link between all the 4GL modules. The runner can be fglnodb because in this program, you do not use any calls to the Informix database interface:

$ fglnodb ex1.42r

If you have compiled the program to C code you should have the following files.

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>The source files</td>
<td>ex1-1.4gl, ex1-2.4gl, ex1-1.per</td>
</tr>
<tr>
<td>The compiled form</td>
<td>ex1-1.42f</td>
</tr>
<tr>
<td>The C code executable</td>
<td>ex1.42e</td>
</tr>
</tbody>
</table>

In this case, you simply run the C code executable:

$ ex1.42e

In either case, you get the same display on the machine fire. Choose the Message box menu item, and then select one of the four icons, enter a title and a message, and then a message box will be displayed.
Running the Example on the Windows Front End

This section describes the complete process for configuring the Windows front end, creating a connection between the Windows machine named `earth` and the UNIX server named `water`, and the execution of the program `ex1`.

Suppose that you are working on the Windows machine `earth` and have finished installing the WTK-client. The first step is to create an `rlogin` connection between `earth` and `water`. To do so, use the WTK-Rlogin emulation. Click the **Add WTK 4GL Connection** icon.
Running the Program on the Front End

The following window is displayed.

![Figure 3-3](image.png)

Enter the following information.

<table>
<thead>
<tr>
<th>Text Box</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer</td>
<td>water</td>
</tr>
<tr>
<td>User</td>
<td>Informix</td>
</tr>
<tr>
<td>Terminal</td>
<td>xterm (This value is automatically set.)</td>
</tr>
<tr>
<td>Commandline</td>
<td>An optional command line to be executed after the connection is successful (For this example, leave it blank.)</td>
</tr>
</tbody>
</table>

You want the terminal to be visible and the login dialog box to be displayed, so leave the connection is visible and the show login-dialog boxes checked. Then validate the information by clicking OK. You now have a new icon named Informix@water. To start the connection, double-click the icon and enter your password when prompted.

The rlogin terminal is now connected to the water server. The necessary environment variables must now be set. Execute the shell script `envf4gl`, created during the installation process and located in the directory where you installed Dynamic 4GL:

```
$ ./envf4gl
```
Running the Program on the Front End

Next, set the FGLSERVER variable to the address of the client machine:

```
$ FGLSERVER=earth:0
$ export FGLSERVER
```

Then check the value of the FGLGUI variable. If it is set to 0 the program is executed in ASCII mode exactly as if compiled with INFORMIX-4GL compilers. If set to 1 it uses the WTK clients and the application is displayed in graphical mode. Check this setting with the following command:

```
$ echo $FGLGUI
```

If it is set to 0 or if it is not set, set it to 1 with the following commands:

```
$ FGLGUI=1
$ export FGLGUI
```

Now that the environment is correctly set, you can start the application. Go to the directory where the program is compiled. If you have compiled it to p-code you should have the following files in the directory:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>The source files</td>
<td>ex1-1.4gl, ex1-2.4gl, ex1-1.per</td>
</tr>
<tr>
<td>The compiled form</td>
<td>ex1-1.42f</td>
</tr>
<tr>
<td>The p-code modules</td>
<td>ex1-1.42m, ex1-2.42m</td>
</tr>
<tr>
<td>The p-code link</td>
<td>ex1.42r</td>
</tr>
</tbody>
</table>

To start the program, just type the name of your runner and, as the first parameter, the name of the file resulting from the link of all the 4GL modules. The runner can be in fglnodb because there are no calls to the Informix database interface in the program:

```
$ fglnodb ex1.42r
```

Immediately after you have started the 4GL program, you will be prompted on the client machine by the following window.
This message only occurs the first time a machine tries to access the communication daemon running on the client machine. In this case, it means that the user Informix on the machine water tries to access earth. You have five seconds to respond before the WTK client answers: no. The choices are:

- **Yes**—The user Informix on the machine water will have access to the local machine for all future connections. The user name and the machine name are recorded in the \windows\rhhosts file.
- **Only once**—The user Informix on the machine water will have access only this one time.
- **No**—The user Informix on the machine water will not have access to the local machine at this time.

For more information, read the “Security Concepts” section in the wtksrv1.wri file, located in the installation directory of the Windows front end.

If you have compiled the program to C code, you should have the following files.

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>The source files</td>
<td>ex1-1.4gl, ex1-2.4gl, ex1-1.per</td>
</tr>
<tr>
<td>The compiled form</td>
<td>ex1-1.42f</td>
</tr>
<tr>
<td>The C code executable</td>
<td>ex1.42e</td>
</tr>
</tbody>
</table>

Next, you simply run the C code executable:

```
$ ex1.42e
```
In either case, you get the same display on the machine earth. Choose the Message box menu item, select one of the four icons, enter a title and a message, and then a message box will be displayed.
Using the Compiler

In This Chapter ........................................... 4-3

Compiling Forms and Help Messages ..................... 4-3
  Compiling Form Specification Files .................... 4-3
  Compiling Help Message Files ......................... 4-4
  Generating a New Database Schema File ............... 4-4

Compiling to P-Code ....................................... 4-5
  Overview of the P-Code Example ....................... 4-5
    Compiling Source Files to Linkable Modules ....... 4-6
    Linking Modules Together to Create P-Code ....... 4-6
  Using C Functions in 4GL Applications Compiled to P-Code ................. 4-7
    Compatibility Between C and Windows NT .......... 4-7
    Linking C Functions with the P-Code Runner ....... 4-7

Compiling to C Code ...................................... 4-15
  Overview of the C Code Example ....................... 4-15
    Compiling Source Files to Linkable Modules ....... 4-16
    Linking Modules to Create C Code Libraries ....... 4-17
  Using C Functions in 4GL Applications Compiled to C Code ................. 4-17
    Compatibility Between C and Windows NT .......... 4-17
    Linking C Functions for Use in C Code Compilations ...... 4-18
In This Chapter

This chapter contains all the needed syntax for compiling 4GL programs into Dynamic 4GL programs. It also explains how to add calls from 4GL programs to C programs and how to make calls from C programs to 4GL programs.

Compiling Forms and Help Messages

This section describes how to compile the form specification files and help messages used by your 4GL applications. It also describes how to generate the database schema file needed to compile 4GL source code, including references to a database.

Compiling Form Specification Files

As with other source code files that compose a 4GL application, you will need to recompile your forms into a Dynamic 4GL-specific format. The compiled form files are used by the application compiled to p-code or to C code.

The tool you use to compile forms is fglform. The extension of the compiled .per files is .42r. The compilation syntax is as follows:

$ fglform formname(.per)

This command compiles the form specification file named formname.per into formname.42r. Note that the .per extension is not mandatory on the command line.

Because you might recompile forms on a machine other than your development machine, the fglform compiler is also available in the runtime package.
Compiling Help Message Files

The second compiler is the help message files compiler, named `fglmkmsg`. This compiler is equivalent to the Informix `mkmessage` compiler.

Following is the syntax for the `fglmkmsg` tool:

```
$ fglmkmsg input_file output_file
```

This command will compile the help message file into a file that the Dynamic 4GL runtime product can use. The following command decompiles the file given as the parameter, and the output is written to the standard output:

```
$ fglmkmsg -r compiled_file
```

Generating a New Database Schema File

No changes are required to the database before using Dynamic 4GL. But a schema file must be generated each time that the database structure changes.

Run the following command in a directory and environment where you have access to your database (that is, you should be able to access the database with the Informix tools INFORMIX-SQL and DB-Access):

```
$ fglschema database_name
```

The file `database_name.sch` is generated. The environment variable `FGLDBPATH` must include the path to the directory containing this file, to allow you to compile 4GL programs using references to the database.

In addition, two other files are generated if needed: `databasename.att` and `databasename.val`. These files are used to manage the tables `syscolatt` and `syscolval`, respectively.
Compiling to P-Code

This section describes how to compile a sample 4GL program to linkable modules and how to link those modules together to create an executable program. It also describes how to use C functions in your applications.

Overview of the P-Code Example

In this section, you will compile the following simple 4GL program named example.4gl:

```
MAIN
  DISPLAY "Hello World"
END MAIN
```

Before executing this simple program, you first need to compile it and then link all the needed modules and the p-code runner together.

Here is the complete compilation schema.

The name of the Dynamic 4GL p-code compiler is fgl2p. With this tool, you can compile the 4GL source code into p-code executables or into libraries.
Overview of the P-Code Example

Compiling Source Files to Linkable Modules

By convention, the following extensions are used for filenames:

- .4gl for the source code files
- .42m for the compiled modules
- .42r for the file resulting from the linking of compiled modules

The syntax for the first step of the compilation, compiling 4GL source code into linkable modules is:

```
$ fgl2p 4gl_source_code.4gl
```

For example:

```
$ fgl2p example.4gl
```

This line compiles the 4GL source code file `example.4gl` to the module `example.42m`.

Linking Modules Together to Create P-Code

Here is the syntax for linking the compiled .42m modules together to create the executable or a library. This link also checks for C functions included in the runner (see the next section) specified by the FGLRUN environment variable:

```
$ fgl2p -o executable.42r module1.42m [module2.42m] ...
```

This line links the compiled modules `module1.42m` and `module2.42m` into the executable.42r. The following line links the compiled modules `module1.42m` and `module2.42m` into the library library.42x:

```
$ fgl2p -o library.42x module1.42m [module2.42m] ...
```

This library can be used as an object module file when linking applications that use calls to functions defined in the library.

The .42m modules are linked together into the .42r hash table containing the cross-references to all functions and variables used in the 4GL application. This means that all unresolved or faulty references (missing functions, function calls with an incorrect number of parameters or return values) are detected at link time instead of at runtime. At runtime only the .42r and .42m modules containing the MAIN section are loaded immediately into memory. All other .42m modules are loaded only when needed.
As an immediate result, every module, and particularly all library modules, will appear only once in a whole application, with beneficial effects on the size of the p-code modules constructing the application.

At runtime all modules linked together must be located in one of the directories specified by the FGLLDPATH environment variable.

**Using C Functions in 4GL Applications Compiled to P-Code**

This section describes a strategy for using C functions in your applications and gives you a step-by-step example.

**Compatibility Between C and Windows NT**

When using C functions in your 4GL applications, you should be aware that these functions can cause problems when you port the application to a platform other than the one used to develop it. These problems especially occur when you try to port an application from UNIX to Windows NT and vice versa. This problem can also occur when you use too many specific calls to system features.

In both cases, try to reduce calls to C functions and system commands to reduce the risk of problems when porting to other platforms.

Dynamic 4GL contains extra functions and features, allowing you to avoid calls to most of the C functions and calls to system features. For a description of the new extensions to the 4GL language, see Chapter 6, “Extensions to the 4GL Language.”

**Linking C Functions with the P-Code Runner**

Because the low-level instruction set is defined in the p-code runner, and because C functions have only a low-level implementation (that is, they do not change the 4GL syntax), they must be linked with the runner at its creation.

To use C functions in a 4GL program, must:

- define the C functions in a C extension file.
- compile your C files and the C extension file.
- build the runner with the C files.
The `fglmkrun` tool allows you to generate a specific runner with C functions. It uses the following syntax:

```
fglmkrun  [-o myrunner]
           [-d ix711]
           [extfile] [ofile1 ...] [lib1 ...]
```

The following table describes the elements of this command.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>myrunner</td>
<td>Name of the specific runner</td>
</tr>
<tr>
<td>extfile</td>
<td>Name of a specific C extension file (default: <code>$FGLDIR/lib/fglExt.c</code>)</td>
</tr>
<tr>
<td>cfile1</td>
<td>Name of a C source file to be compiled and linked with the runner</td>
</tr>
<tr>
<td>ofile1</td>
<td>Name of a C object file to be linked with the runner</td>
</tr>
<tr>
<td>lib1</td>
<td>Name of a C library to be linked with the runner</td>
</tr>
</tbody>
</table>

You must specify the `-d ix711` option if programs are to run with Informix 7.x database servers, or, alternatively, you can set the environment variable `FGLDBS` to `ix711`.

Use the C extension file is designed to specify the name, the number of parameters, and the number of values returned from every C function linked with the runner.

This is the standard extension file `$FGLDIR/lib/fglExt.c`:

```
#include f2c/fglExt.h
UsrData usrData[2]=
( 0, 0 )
):
UsrFunction usrFunctions[2]=
(0,0,0,0 )
):
```

The two arrays `usrData` and `usrFunctions` must always be present in the file. The last record of each array should be a line with all the elements set to 0. The `usrData` array contains the name of the global variables modified by your C programs, and `usrFunctions` contains the name of the C functions called from the 4GL modules.
You can copy the file $FGLDIR/lib/fglExt.c and adapt it to your own needs. For example:

```c
#include f2c/fglExt.h
#include f2c/r_c.h
int my_func1_cname(int nargs);
int my_func2_cname(int nargs);
UsrFunction usrfuntions[]=
{ "my_func1_4gname",my_func1_cname,my_func1_nbparam,my_func1_nbret },
{ "my_func2_4gname",my_func2_cname,my_func2_nbparam,my_func2_nbret },
{ 0,0,0,0 }
);
```

The following table describes the elements in this example.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_func1_4glname</td>
<td>Function name in the 4GL program</td>
</tr>
<tr>
<td>my_func1_cname</td>
<td>Function name in the C module</td>
</tr>
<tr>
<td>my_func1_nbparam</td>
<td>Number of parameters (-1 means variable)</td>
</tr>
<tr>
<td>my_func1_nbret</td>
<td>Number of return values (-1 means variable)</td>
</tr>
</tbody>
</table>

This first example is a simple call to a C function in a 4GL module.

First create your C file (examplec.c):

```c
#include stdio.h
int fncc1(int n)
{
    printf("This a C file.");
    return 0;
}
```

Compile it with your C compiler:

```
$ cc -c example.c
```

Before any modification, copy fglExt.c into your working directory to make it available for all users.
Using C Functions in 4GL Applications Compiled to P-Code

After that, edit `fglExt.c` and update it with the following definitions:

```c
#include f2c/fglExt.h
UsrData usrData[] = {
    ( 0, 0 )
};
int fncc1(int n);
UsrFunction usrFunctions[] = {
    { fncc1, fncc1, 0, 0 },
    { 0, 0, 0, 0 }
};
```

Now you can build the new runner with the following command:

```
$ fglmkrun -o newrunner example.o fglExt.c
```

This command builds a runner (the link between the Informix libraries, your system libraries, the Dynamic 4GL libraries, and the file `example.o`) named `newrunner`. This runner is for 4.x Informix databases. If you have a 6.x or later version of ESQL/C, add the flag `-d ix711` to the previous command.

Do not give your new runner the same name as one of the files located in the current directory. When you have created the new runner, you can create the 4GL example (`example.4gl`) with the following lines:

```4gl
MAIN
CALL fncc1()
END MAIN
```

Compile the `.4gl` file with the following command:

```
$ fg12p -c example.42m example.4gl
```

And link your object file `example.42m` to `example.42r` with the following commands:

```
$ FGLRUN=newrunner
$ export FGLRUN
$ fg12p -o example.42r example.42m
```

The shell script `fg12p` uses the value of the `FGLRUN` variable to determine which runner to link with. If you do not set the `FGLRUN` variable before linking your 4GL program, the compiler will generate an error because the `fncc1` function was undefined.

Now you can execute your p-code executable with the following command:

```
$ newrunner example.42r
```
Using C Functions in 4GL Applications Compiled to P-Code

Note that with SCO systems, the use of `fglmkrun` during a manual installation causes the error message:

```
Symbol not found fileno
First referenced in file.../lib/libf2c.a
```

This problem is due to differences between the different versions of the SCO libraries. The solution to this problem is to create a file named `fileno.c` containing the following lines:

```c
#include stdio.h
#undef fileno
int fileno(f)
    FILE *f ;
    { return(f->__file) ; }
```

Then execute `fglmkrun` with `fileno.c` as an additional parameter (for Informix 5.x):

```
$ fglmkrun -o fglrun fileno.c $FGLDIR/lib/fglExt.c
```

Building on what you know about calling a C function from a 4GL module, you can call 4GL out of a C function. Use the `fCall` function in your C programs:

```
fCall("funcname",nbparam)
```

where `funcname` is the name of the 4GL function to call (CHAR), and `nbparam` is the number of parameters (INTEGER). This function returns the number of return values (INTEGER).

The parameters must be pushed on the stack before the call, and the return values must be popped from the stack after returning. The 4GL function must be declared external in the C extension file. Update the C file with the following statements:

```c
#include stdio.h
#include f2c/fglExt.h
int fncc1(int n)
{
    fCall("fnc2",0);
    return 0;
}
```
Compile this using the following command:

```bash
$ cc -c example.c -I$FGLDIR/include
```

Now update file `fglMyExt.c`:

```c
#include f2c/fglExt.h
UsrData usrData[]={
    ( 0, 0 )
};
extern int fnc1(int n);
UsrFunction usrFunctions[]={
    {"fnc1", fnc1, 0, 0 },
    {0, 0, 0, 0 }
};
```

Then build the new runner with the following command line:

```bash
$ fglmkrun -o newrunner example.o fglMyExt.c
```

Then update the 4GL example:

```4gl
MAIN
CALL fncc1()
END MAIN
FUNCTION fnc2()
DISPLAY "You are in 4gl function"
END FUNCTION
```

Compile it with `fgl2p`:

```bash
$ FGLRUN=newrunner
$ export FGLRUN
$ fgl2p -c example.42m example.4gl
$ fgl2p -o example.42r example.42m
```

Now you can run it with the new runner:

```bash
$ newrunner example.42r
```

The last step is to modify 4GL global variables in C functions. Every variable must be defined in the C file as well as in the C extension file used to build the specific runner. The C extension file also contains the definitions of the C functions.

The global 4GL variables are internally redefined, so you have to use the `CNAME` macro to reference them in your C files containing your C functions.
Furthermore, every variable must be defined as external to the C module with its corresponding type. Use the following syntax:

```c
#define variable_name_in_4gl CNAME(variable_name_in_4gl)
```

where `variable_name_in_4gl` is the name of the variable in 4GL.

Here is an example of a file containing the C functions called from the 4GL:

```c
#include stdio.h
#include f2c/fglExt.h
#define var CNAME(var)
#define res CNAME(res)
extern int var;
extern char res[101];
int fncc1(int n)
{
    printf("%s %d\n", res, var);
    return 0;
}
```

Compile the C file using the following command:

```bash
$ cc -c example.o -I$FGLDIR/include
```

Now modify the file `fglMyExt.c`. Use the `GLOB_type` macro to create the relationship between the name of the global variable in C and the one in 4GL:

```c
GLOB_type(varname[. varlength]);
```

The following table describes the elements of this command.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>The type of the variable</td>
</tr>
<tr>
<td>varname</td>
<td>Name of the variable as defined in 4GL</td>
</tr>
<tr>
<td>varlength</td>
<td>Length of the variable as defined in 4GL (only for CHAR variables)</td>
</tr>
</tbody>
</table>
Using C Functions in 4GL Applications Compiled to P-Code

Example:

```c
#include f2c/fglExt.h
GLOB_CHAR(res,100);
GLOB_INT(var);

UsrData usrData[]={
    GLOB(var),
    GLOB(res),
    { 0, 0 }
};
int fnc1(int n);
UsrFunction usrFunctions[]={
    { "fnc1", fnc1, 0, 0 },
    {0,0,0,0 }
};

Create the new runner:

$ fglmkrun -o newrunner example.o fglExt.c
```

The supported types are:

<table>
<thead>
<tr>
<th>Type</th>
<th>4GL Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>GLOB_CHAR</td>
</tr>
<tr>
<td></td>
<td>GLOB_VARCHAR</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>GLOB_SMALLINT</td>
</tr>
<tr>
<td>INTEGER</td>
<td>GLOB_INT</td>
</tr>
<tr>
<td>SMALLFLOAT</td>
<td>GLOB_SMALLFLOAT</td>
</tr>
<tr>
<td>FLOAT</td>
<td>GLOB_FLOAT</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>GLOB_DECIMAL</td>
</tr>
<tr>
<td>MONEY</td>
<td>GLOB_MONEY</td>
</tr>
<tr>
<td>DATE</td>
<td>GLOB_DATE</td>
</tr>
</tbody>
</table>

The list of types supported can be also found in the file `$FGLDIR/include/f2c/fglExt.h`.

Global RECORD and ARRAY statements are not allowed.
Using the Compiler

Compiling to C Code

Adapt your 4GL example as follows:

```plaintext
GLOBALS
    DEFINE var INTEGER,
    res CHAR(100)
END GLOBALS
MAIN
    LET var = 15
    LET res = "The result is 
    CALL fncc1()
END MAIN
```

Compile it with `fgl2p`, and run it with the new runner:

```
$ fgl2p -o example.42r example.4gl
$ newrunner example.42r
```

Compiling to C Code

You can compile 4GL programs to C code. However, Informix strongly advises you to use p-code compilation rather than C code. Although p-code executes more slowly than C code, the difference is not significant. With C code compilation, you must recompile the whole program each time you change the execution platform, whereas with p-code, you only need to rebuild your runner.

Overview of the C Code Example

In this section, you compile the following simple 4GL program named `example.4gl` to C code:

```plaintext
MAIN
    DISPLAY "Hello World"
END MAIN
```

Before executing this simple program, you first need to compile it and then link all the needed modules.
Overview of the C Code Example

Here is the complete compilation schema.

The name of the Dynamic 4GL C code compiler is \texttt{fgl2c}. With this tool you can compile the 4GL source code into C code executables or into libraries.

\textbf{Compiling Source Files to Linkable Modules}

The following conventions are used for the filename extensions:

- \texttt{.4gl} for the source code files
- \texttt{.42o} for the compiled modules
- \texttt{.42e} for the file resulting from the linking of compiled modules, system libraries, and p-code libraries

Here is the syntax for the first step of the compilation, compiling 4GL source code into linkable modules:

\begin{verbatim}
$ fgl2c -c 4gl_source_code.4gl
\end{verbatim}

For example:

\begin{verbatim}
$ fgl2c -c example.4gl
\end{verbatim}

This line compiles the 4GL source code file \texttt{example.4gl} to the module \texttt{example.42o}. 

**Using C Functions in 4GL Applications Compiled to C Code**

**Linking Modules to Create C Code Libraries**

Here is the syntax for linking the compiled .42o modules together to create the executable:

```
$ fg12c -o executable.42e module1.42o [module2.42o] ...
```

This line links the compiled modules `module1.42o` and `module2.42o` into the `executable.42e`.

The procedure to create C code libraries is a little different from that for creating p-code libraries. To build the C code libraries, you must use the `ar42o` tool. The syntax of `ar42o` is:

```
$ ar42o libname.a module1.42o [module2.42o] ...
```

This line uses the UNIX `ar` command to create the library named `libname.a` made of the compiled modules `module1.42o` and `module2.42o`. This library can be used as an object module file when linking applications that use calls to functions defined in the library. For more information on `ar`, read the `ar` man page on your UNIX system.

**Using C Functions in 4GL Applications Compiled to C Code**

This section describes a strategy for using C functions in your applications and gives you a step-by-step example.

**Compatibility Between C and Windows NT**

When using C functions in your 4GL application, you should be aware that these functions can cause problems when you port the application to a platform other than the one used to develop it. These problems occur most often when you try to port an application from UNIX to Windows NT and vice versa. This problem can also occur when you use too many specific calls to system features.

In both cases, try to reduce calls to C functions and system commands to reduce the risk of problems when porting to other platforms.

Dynamic 4GL contains extra functions and features, allowing you to avoid calls to most of the C functions and calls to system features. For a description of the new extensions to the 4GL language, see Chapter 6.
Using C Functions in 4GL Applications Compiled to C Code

**Linking C Functions for Use in C Code Compilations**

With C code, the C functions are linked in the same manner as any other modules during the application link phase. You just have to follow a few rules to successfully call C functions from 4GL applications and vice versa.

With C code, in order to call a C function from a 4GL application, you do not need a C extension file to create the relationship between the name of the C function and the name of the 4GL function. But you have to call a macro named CFACE defined in the f2c/r_c.h header file. This means that you will also have to include this file at the beginning of your C files. Here is the syntax:

```c
CFACE (C_function_name, parameters_number, returned_values_number)
```

The following table describes the elements of this command.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_function_name</td>
<td>Name of the C function</td>
</tr>
<tr>
<td>parameters_number</td>
<td>Number of parameters transmitted to the function</td>
</tr>
<tr>
<td>returned_values_number</td>
<td>Number of values returned by the function</td>
</tr>
</tbody>
</table>

Use this macro for all C functions called from 4GL applications. Include the f2c/r_c.h header file in all the C files that call this macro.

If you want to use the same source files to compile your applications, either with Dynamic 4GL or with the INFORMIX-4GL compilers, use conditional compiling, as in the following example:

```c
#ifdef Informix
#include f2c/r_c.h
#endif
```

Here is an example of a 4GL application that calls a C function named `mainc`. Following is the 4GL source code, `exCCode.4gl`:

```
MAIN
CALL mainc()
END MAIN
```

Here is the C file, exc.c:

```c
#include f2c/r_c.h /* This is the Informix header file defining the CFACE macro */
#include stdio.h
int mainc(int n)
  printf (*hello from C !!*);
  return 0;
}
CFACE(mainc,0,0)   /* Macro needed for every function call from 4GL */
```

Now compile the two previous files with the commands:

```
$ cc -c exc.o exc.c -D Informix -I$FGLDIR/include
$ fgl2c -c exCCode.42o exCCode.4gl
```

Next, link the compiled modules, the system libraries, and the Informix development libraries together with the `fgl2c` shell script:

```
$ fgl2c -o exc.42e exc.o exCCode.42o
```

To call 4GL functions from a C function, use the `FGLCALL` function in your C functions. This function is also defined in the `f2c/r_c.h` Dynamic 4GL header file. Here is the syntax:

```
FGLCALL(4GL_function_name, parameters_number, returned_values_number)
```

The following table describes the elements of this command.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4GL_function_name</td>
<td>Name of the 4GL function</td>
</tr>
<tr>
<td>parameters_number</td>
<td>Number of parameters transmitted to the function</td>
</tr>
<tr>
<td>returned_values_number</td>
<td>Number of values returned by the function</td>
</tr>
</tbody>
</table>

Here is an example of a 4GL function that calls a C function, which, in turn, calls another 4GL function. This example is made of two 4GL modules and one C file.
Using C Functions in 4GL Applications Compiled to C Code

The first 4GL module is exCCode.4gl:

```4GL
MAIN
DEFINE word CHAR(60)
OPEN WINDOW w1 AT 1,1 WITH 20 ROWS, 50 COLUMNS
ATTRIBUTES(BORDER)
LET word = "How are you?"
CALL mainc(word)
SLEEP 3
CLOSE WINDOW w1
END MAIN
```

The second 4GL module is fnCCode.4gl:

```4GL
FUNCTION fncc1(word)
DEFINE word CHAR(60)
IF word = "How are you?" THEN
    DISPLAY "Very fine and you?" AT 10, 1
END IF
END FUNCTION
```

The C file is exc.c:

```c
#ifdef Informix
#include f2c/r_c.h
#else
#include stdio.h
#include "fncc1.c"
int mainc (int n) {
    CHAR word[13];
    popquote(word, 13);
    pushquote(word, 13);
    FGLCALL(fncc1, 1, 0);
    return 0;
}  
CFACE(mainc, 1, 0)
#endif
```

Note that the C statements popquote, pushquote, pop[...] and push[...] are working exactly as with Informix compilers.

Next, compile these three files:

```
$ cc -c exc.o exc.c -D Informix -I $FGLDIR/include
$ fg12c -c exCCode.4gl
$ fg12c -c fnCCode.4gl
```

Link the three object modules, the system libraries, and the Informix libraries together:

```
$ fg12c -o exCCode.42e exCCode.42o fnCCode.42o exc.o
```
Run the example by typing:

$ exCCode.42e

The next step is to share global variables between C functions and 4GL functions. The definition process for global variables is exactly the same as when you compile your program in C code or in p-code, except that no C extension file is needed. Here is the syntax of the CNAME macro:

```
#define variable_name_in_4gl CNAME(variable_name_in_4gl)
```

where `variable_name_in_4gl` is the name of the variable in 4GL.

To illustrate this macro with C code compilation, you simply modify the previous example to use a global variable instead of a parameter to exchange the data between the 4GL functions and the C function.

Here is the first new 4GL module, `exCCode.4gl`:

```
GLOBALS
  DEFINE word CHAR(12)
END GLOBALS
MAIN
  OPEN WINDOW w1 AT 1,1 WITH 20 ROWS, 50 COLUMNS
  ATTRIBUTES(BORDER)
  LET word = "How are you?"
  CALL mainc()
  SLEEP 3
  CLOSE WINDOW w1
END MAIN
```

The second one is `fnCCode.4gl`:

```
GLOBALS
  DEFINE word CHAR(12)
END GLOBALS
FUNCTION fncc1()
  IF word = "How are you?" THEN
    DISPLAY "Very fine and you?" AT 10, 1
  END IF
END FUNCTION
```
Using C Functions in 4GL Applications Compiled to C Code

The new C function is `exc.c`:

```c
#include <stdio.h>
extern char word[13];
int mainc(int n) {
    printf("\n", word);
    FGLCALL(fncc1, 0, 0);
    return 0;
}
CFACE(mainc, 0, 0)
```

Now compile these three files:

```
$ cc -c exc.o exc.c -D Informix -I$FGLDIR/include
$ fgl2c -c exCCode.4gl
$ fgl2c -c fnCCode.4gl
```

Next, link the three object modules, the system libraries, and the Informix libraries together:

```
$ fgl2c -o exCCode.42e exCCode.42o fnCCode.42o exc.o
```
Differences Between 4GL and Dynamic 4GL Compilers

In This Chapter .............................................................................. 5-3
Differences Between 4GL and Dynamic 4GL Compilers .................. 5-3
   The Dynamic 4GL Configuration Files ........................................ 5-3
   List of Differences ..................................................................... 5-4
4GL Extensions ............................................................................. 5-6
In This Chapter

Dynamic 4GL is based on Version 4.1 of INFORMIX-4GL. Most of the time, however, it is possible to customize the behavior of the compiler so that it can compile programs originally created with different versions of INFORMIX-4GL.

Differences Between 4GL and Dynamic 4GL Compilers

The differences between the INFORMIX-4GL compiler and the Dynamic 4GL compiler can be controlled by the Dynamic 4GL configuration file $FGLDIR/etc/fglprofile.

The Dynamic 4GL Configuration Files

To control the behavior of the Dynamic 4GL compiler, use a configuration file located in the $FGLDIR/etc directory. This file, fglprofile, contains a list of entries that can be changed to control the behavior of the compiler. The complete list of entries that can be set in this file is described in Appendix B, “The Configuration File.”

There are three configuration files. The main configuration file is the fglprofile file located in the $FGLDIR/etc directory. This configuration file is loaded first and is loaded each time the compiler is used.

The second configuration file is the one specified by the FGLPROFILE environment variable. If this variable is set in the environment of the current user, the corresponding file is loaded after the fglprofile file. All entries defined in the two files are set to the value defined in the latest loaded configuration file.
The third configuration file is the one located in the directory defined in one of the two previous configuration files with the entry named fglrun.default.

**Important:** This configuration file must have the same name as the 4GL application.

**List of Differences**

**Initialization of variables.** The 4GL compiler initializes global variables with empty structures, whereas the Dynamic 4GL compiler initializes them to **null**.

The 4GL compiler initializes decimal variables to 0, and the Dynamic 4GL compiler sets them to **null**, like INFORMIX-4GL RDS (Rapid Development System).

**Datetime.** With the instruction CURRENT, the Dynamic 4GL compiler manages three positions of the fraction part, whereas 4GL manages only two.

**Arrays.** If you call for an index that is out of range of an array, the compiler can either stop execution of the program with a runtime error, or it can return NULL for the non-existing elements of an array and continue with the program. Use the entry `fglr.run.arrayIgnoreRangeError` in the configuration files if you want the compiler to continue execution.

**Input array.** If you use an INPUT or INPUT ARRAY instruction, be sure that at least one field is not declared as **noentry**.

Following is an example of an INPUT ARRAY instruction where Informix accepts all fields declared as **noentry**:

```
INPUT ARRAY rec_array WITHOUT DEFAULTS FROM scr_arr.*
BEFORE ROW
EXIT INPUT
END INPUT
```

**Mouse usage.** Graphical applications make it possible to use the mouse to move from one field to another directly, without passing by an intermediate field in the INPUT statement. Therefore, an entry exists in the configuration file that allows you to execute all the intermediate triggers when users move from one field to another. This entry is named `dialog.fieldOrder`. 
Another mouse-related issue is right-clicks and double-clicks. The following two entries in the configuration files accommodate these mouse actions:

```plaintext
gui.key.doubleClick.left = "key" (default KEY_accept)  
gui.key.click.right     = "key" (default F36)
```

**Reports.** It is possible to choose the result of an aggregate function (avg,sum,...) used in reports, when the result of this function is NULL. It can either return 0 or NULL. To choose the correct behavior, use the `report.aggregateZero` entry in the configuration file.

Also in reports, it is possible to use a global variable to define the PAGE LENGTH of a report. Simply set this variable to the desired value before calling the START REPORT statement.

**Cursors scope range.** With Informix 7.x servers, you can choose the scope range for cursors at compile-time. By default the cursor scope is local to the module, but it can be defined to be global to the application. To configure this behavior, use the `fglruncursor.global` entry in the configuration file. This choice is made at runtime with Dynamic 4GL.

**Menu.** The 4GL menu can be displayed either on the top of the application window or on the right side of the screen on top of the hot key buttons. Use the `menu.style` entry in the configuration file to choose the position of the menu in the application window.

It is also possible to have a bitmap displayed on menu buttons. To do so, precede the label by the character @; for example:

```plaintext
menu "blabla"  
  command "@stop.bmp"  
  exit program  
  command "hello"  
  exit menu  
end menu
```

**The sqlexit statement.** The `sqlexit()` function provides a way of terminating the `sqlexec` process. It must be invoked as follows:

```plaintext
CALL --#sql:  
sqlexit()
```

You must restart the `sqlexec` process before the next SQL statement. Use the following statement:

```plaintext
DATABASE databasename
```
PROMPT. The 4GL PROMPT statement waits for an answer in the graphical window at the prompt line if the graphical window is opened first. If there is no graphical window open before the PROMPT statement is executed, the prompt is made in the ASCII terminal; for example:

```plaintext
MAIN
DEFINE C CHAR(1)
DISPLAY « Hello »
PROMPT « Press any key » FOR CHAR C
END MAIN
```

In this example, PROMPT is executed in the terminal and in terminal mode, even if you are in graphical mode.

The following provides an example of how to execute PROMPT in graphical mode:

```plaintext
MAIN
DEFINE C CHAR(1)
DISPLAY « Hello » AT 5,5
PROMPT « Press any key » FOR CHAR C
END MAIN
```

4GL Extensions

Here is a summary of the features added to the INFORMIX-4GL language. Most of these features have been added to avoid the use of calls to external C functions and to improve the language. These extensions are described in greater detail in Chapter 6, “Extensions to the 4GL Language.”

Channel functions, a way to use I/O streams. Channel extensions provide access to the files and the processes of the system without using the RUN statement. Your application requires fewer resources than with the RUN statement and allows you to communicate with pipes with other applications.

DDE functions, to call a Windows application from 4GL. With this new extension, you can invoke a Windows application and send or receive data to or from it. To use this new functionality, the program must be executed on a Windows machine, or on a UNIX machine but from a Windows front end.
Starting a system application using the fgl_system() function. You can run a program using a UNIX terminal to display the output. Even if the running Dynamic 4GL program has been started without a visible terminal with the Windows front end, the UNIX terminal will be started and placed in the foreground during the execution of the external program. Then it will be placed in the background when the program using it is finished.

Drawing in 4GL with the canvas functions. This set of functions allows you to draw basic shapes in an area defined like an array.

Retrieving the key pressed using fgl_getkey(). This function waits for a keystroke and returns the key code of a pressed key.

Field functions. These functions have been added to manage fields from the 4GL source code. You can, for example, find out the name of a field, set or get the value dynamically, and set or get the cursor position in a field.

Window functions. Like the field functions, these functions have been added to manage the different windows in your 4GL applications. You can use them to find out the name of the current window, its size, and other characteristics.

New form specification and function. These new specifications add several features. Several specifications run only in graphical mode, such as check boxes, radio buttons, buttons, and .bmp fields. Some others allow you to manage fields from the form, such as scrolling fields, no list attributes, key definitions, and drawing attributes.

New 4GL dialog box functions. You can create different types of dialog boxes. You can add a title and window size independently from your source specification. You can also draw items or dynamically change the labels on buttons.


Toolbars. A toolbar can be added to the top of the screen to represent frequently used commands.

The report pager. A pager has been written that allows you to scroll reports that appear on the screen. For wide reports of more than 80 columns, you can also scroll horizontally. The correct sizing of the vertical scrollbar requires a PAGE TRAILER. Switch to the previous or next page by clicking a button.
4GL Extensions

To manage interruption of the display, the **int_flag** has to be tested after every **OUTPUT TO REPORT** instruction.

The same pager can be used from the UNIX prompt:

```bash
$ pager.42e [filename]
```

for the C version or

```bash
$ fglpager [filename]
```

for the p-code version or on Windows NT.

The **fglpager** command has the same functionality as **pager.42e**, except:

- you can only scroll 10 pages backwards.
- you can see all pages, but you must specify a database name as parameter `-d dbname` because the page uses temporary tables. Reports in 4GL programs can also use temporary tables. Because they use a lot of space, you must first call the function `fgl_report_use_temptable()` to enable them. Otherwise, you can see only 10 pages backwards.
- If you execute the report with `FGLGUI=0`, the pager will display all the report without stopping after each page.

**Screen record without size.** With **fglform (.per compiler)**, you are not required to specify the screen record size, but if you do not specify the size, you will not have a scrollbar.

**Character filter.** You can define conversion files to be used for characters on the GUI.
Extensions to the 4GL Language

In This Chapter ............................................ 6-5

Non-Graphical 4GL Extensions .................................. 6-5
  Channels: Handling Input/Output Streams ................. 6-5
    USE channel — Initialization of Channel Extensions .... 6-6
    channel::Open_file() — Opens a File ................. 6-6
    channel::open_pipe() — Opens a Pipe ................. 6-7
    channel::set_delimiter() — Sets the Default Separator .. 6-8
    channel::read() — Reads Data from an Opened Channel .. 6-8
    channel::write() — Writes Data to a Pipe or Stream ... 6-9
    channel::close() — Closes the Channel ............... 6-10
    Channel Error Code .................................... 6-10

DDE Communication Protocol .................................. 6-10
  Using the DDE Extension .................................. 6-11
    DDEConnect() — Opens a DDE Connection ............... 6-12
    DDEExecute() — Executes a Command Using DDE ........ 6-12
    DDEPoke — Transmits Values to a Windows Program ..... 6-13
    DDEPeek — Gets Values from a Windows Program ....... 6-13
    DDEFinish — Closes a DDE Connection ................. 6-14
    DDEFinishAll — Closes All DDE Connections .......... 6-14
    DDEGeterror — Error Manager Function ................. 6-14
    Supported Versions of Windows Applications ............ 6-15

Extension of DISPLAY ARRAY Command ....................... 6-15

Other Non-Graphical Extensions. .............................. 6-16
  fgl_getkey() ............................................. 6-17
  uiInkey() - Getkey in C Functions ....................... 6-21
  fgl_system() ............................................. 6-21
  WinExec() and WinExecWait() ............................. 6-22

New Form Specification Fields ................................ 6-23
  List Boxes .............................................. 6-23
  Buttons .................................................. 6-25
Configuration Functions ........................................ 6-60
Drawing Functions ............................................. 6-62
Mouse Management Functions ................................. 6-65
Toolbars ......................................................... 6-66
In This Chapter

This chapter describes the new 4GL language extensions added by Dynamic 4GL. The new features are divided into five categories:

- 4GL functions that are not related to visual effects, such as the input/output functions.
- New features that can be added to the form specification files.
- 4GL functions that produce visual effects in 4GL applications.
- The drawing capabilities of Dynamic 4GL.
- The ability to include toolbars in your application.

Non-Graphical 4GL Extensions

This section describes the 4GL extensions for:

- handling input/output streams.
- using Dynamic Data Exchange.
- extending the DISPLAY ARRAY command.

This section also describes the `fgl_getkey()`, `fgl_system()`, and `WinExec()` functions.

Channels: Handling Input/Output Streams

Channel extensions provide access to the system, the files, and the processes, without using the RUN statement. With this set of functions, your application requires fewer resources than with the RUN statement and allows you to communicate with pipes with other applications.
Channels: Handling Input/Output Streams

All the functions, except **USE channel**, are prefixed by `channel::` to indicate that they belong to the `channel` class.

In the section “DDE Communication Protocol” on page 6-10, you will find a complete example illustrating the use of the channel functions and DDE functions.

**USE channel — Initialization of Channel Extensions**

Syntax: 

```
USE channel
```

Returns: None

This statement tells the compiler that channel extensions will be used during the execution of the 4GL program. This statement must be located before the MAIN clause in the source code. For example:

```
USE channel
MAIN
END MAIN
```

**channel::Open_file() — Opens a File**

Syntax: 

```
channel::open_file(handle, filename, oflag)
```

Returns: None

- **handle**: `CHAR(xx)` Unique identifier for the specified filename
- **filename**: `CHAR(xx)` Name of the file you want to open
- **oflag**: `CHAR(1)`
  - `r` Read mode (standard input if the filename is empty)
  - `w` Write mode (standard output if the filename is empty)
  - `a` Append mode: writes at the end of the file (standard output if the filename is empty)
  - `u` Reads standard read/write on standard input (filename must be empty)
This function opens the file specified by `filename` and prepares the file for reading or writing, as specified by `oflag`. The filename is assigned to the handle that will be called for the different operations on the opened channel. For example:

```plaintext
CALL channel::open_file("stream", "fglprofile", "r")
```

**channel::open_pipe() — Opens a Pipe**

**Syntax**

```plaintext
channel::open_pipe(pipe_handle, command, oflag)
```

- `pipe_handle` CHAR(`xx`) Unique identifier for the specified command
- `command` CHAR(`xx`) Name of the command you want to execute
- `oflag` CHAR(1) 
  - `r` Read mode
  - `w` Write mode
  - `a` Append mode: writes at the end of the file
  - `u` Read and write from command (only available for the UNIX system)

**Returns** None

This function opens the pipe specified by `command` and prepares the pipe for reading or writing, as specified by `oflag`. The command is assigned to the handle called for the different operations on the opened channel. For example:

```plaintext
CALL channel::open_pipe("pipe", "ls -l", "r")
```
Channels: Handling Input/Output Streams

channel::set_delimiter() — Sets the Default Separator

Syntax:

```
CALL channel::set_delimiter(handle, delimiter)
```

- **handle**
  - Type: CHAR(xx)
  - Unique identifier for open channel

- **delimiter**
  - Type: CHAR(1)
  - Delimiter of field

Returns: None

Because channel read/write functions are the same as those used by LOAD/UNLOAD functions, the default separator is defined by the `DBDELEMITER` environment variable. The default value is the `|` (pipe) character. This function allows you to change the delimiter of each opened channel defined by its handle within a 4GL program. If `delimiter=""` (empty string), no delimiter is used. For example:

```
CALL channel::set_delimiter("pipe","."")
```

channel::read() — Reads Data from an Opened Channel

Syntax:

```
channel::read(handle, buffer-list)
```

- **handle**
  - Type: CHAR(xx)
  - Unique identifier for open channel

- **buffer-list**
  - List of variables, if you use more than one variable, you must enclose the list in brackets ([ ])

Returns: SMALLINT

TRUE if data has been read from handle; FALSE if an error occurs

This function reads data from the stream specified by the handle and stores the data in a buffer. The storage buffer can be a single variable, a simple array, or a record.

**Warning:** Specifying a constant value as `buffer-list` is not detected at compile time and will generate a core dump on UNIX machines and a general protection fault on Windows systems.

Following are some examples of this function. Here is a `read` function return value in a variable buffer:

```
DEFINE buffer CHAR(128)
CALL channel::read("pipe_handle", buffer) RETURNING ret
```
Here is a `read` function returning data in a simple array:

```plaintext
DEFINE buffer ARRAY[1024] of CHAR(128)
DEFINE I INTEGER
LET I = 1
WHILE channel::read("pipe_handle", buffer[I])
  LET I = I + 1
END WHILE
```

Here is a `read` function returning data in a record:

```plaintext
DEFINE buffer RECORD
  Buff1 CHAR(128),
  Buff2 CHAR(128),
  Buff3 INTEGER
END RECORD
CALL channel::read("handle", [buffer.Buff1, buffer.Buff2, buffer.Buff3])
```

### `channel::write()` — Writes Data to a Pipe or Stream

**Syntax**

`channel::write(handle, buffer_list)`

- **handle**
  - `CHAR(xx)`
  - Unique identifier for open channel

- **buffer_list**
  - List of variables; if you use more than one variable, you must enclose the list in brackets (`[]`)...

**Returns**

None

This function writes data from a stored buffer to a stream. The storage buffer can be a single variable, a simple array, a record, or a string between double quotes (`"`). For example:

```plaintext
CALL channel::write("handle", "hello world")
```
**DDE Communication Protocol**

**channel::close() — Closes the Channel**

**Syntax**

```
channel::close(handle)
```

**handle**

CHAR(xx) Unique identifier for open channel

**Returns**

None

This function closes the channel specified by `handle`. For example, assume `handle` is called `handle1`:

```
CALL channel::close("handle1")
```

**Channel Error Code**

Even though several channel functions return no error code, you can test the status of the called function like all other 4GL functions. You can see the different error codes returned by testing the `status` variables:

- 2001: Unsupported mode for 'open file'.
- 2002: Cannot open pipe.
- 2003: Unsupported mode for 'open pipe'.
- 2004: Cannot write to unopened file or pipe.
- 2005: Channel write error.
- 2006: Cannot read from unopened file or pipe.

**DDE Communication Protocol**

Dynamic Data Exchange (DDE) is a form of interprocess communication that uses shared memory to exchange data between applications. Applications can use DDE for one-time data transfers and for ongoing exchanges in applications that send updates to one another as new data becomes available.

With this new extension, you can invoke a Windows application and send or receive data to or from it. To use this new functionality, the program must be executed on a Windows PC, or on a UNIX workstation but only from the Windows front end.
The DDE extension might not run with some of the latest Microsoft Office versions (such as Office 97) because these applications do not fully support DDE. Please refer to the documentation for the corresponding software.

At the end of this section, you will find a complete example illustrating DDE and the channel features.

**Using the DDE Extension**

The DDE 4GL process is a four-part process. The 4GL application sends to the Windows front end (WTK client) the DDE order using the TCP/IP channel (1). The Windows front end executes the DDE order using the TCL/TK functions and sends the data to the Windows application through the DDE communication process (2). The Windows application executes the command and sends the result, which can be data or an error code, to the Windows front end (3). The Windows front end sends the result to the 4GL application using the TCP/IP channel (4).

To start a Windows application on the client side, use the `winexec` function.
DDE Communication Protocol

DDEConnect() — Opens a DDE Connection

Syntax

DDEConnect(progname, docname)

progname     CHAR(128)  Program name

docname      CHAR(128)

Returns

TRUE if the connection has been successfully opened; FALSE if an error occurs (The error can be seen using the DDEGeterror function.)

A DDE connection is represented by a unique identifier consisting of a program name followed by a topic that can be a working document or system. For example:

CALL DDEConnect("EXCEL", "Document1")

DDEExecute() — Executes a Command Using DDE

Syntax

DDEExecute(progname, docname, command)

progname     CHAR(128)  Program name

docname      CHAR(128)  Working document or system

command      CHAR(2048) Command to executed through DDE (The syntax of the command depends on the calling program.)

Returns

TRUE if the command has been successfully executed; FALSE if the command has encountered an error (You see the error using the DDEGeterror function.)

This function executes a command in the specified program using the DDE channel. This program can be a macro or any other command available in the calling program. For example:

LET command = "EXECUTE(\\"macro1.xlm!Save1\\":FALSE)"
CALL DDEExecute("EXCEL", "Document1", command ) RETURNING ret
**DDEPoke — Transmits Values to a Windows Program**

**Syntax**

CALL DDEPoke(progname, docname, cells, values)

- **progname**  
  CHAR(128)  
  Program name

- **docname**  
  CHAR(128)  
  Working document or system

- **cells**  
  CHAR(128)  
  Working items

- **values**  
  CHAR(128)  
  Data sent to the progname

**Returns**

TRUE if the values have been successfully transmitted; FALSE if an error occurs (The error can be seen using the function DDEGeterror.)

**DDEPoke** sends data to the specified program and document using the DDE channel. For example:

```sql
LET val="12\t13\t14"
CALL DDEPoke("EXCEL", "Document1", "R1C1: R2C2", val) RETURNING ret
```

**DDEPeek — Gets Values from a Windows Program**

**Syntax**

CALL DDEPeek(progname, docname, cells)

- **progname**  
  CHAR(128)  
  Program name

- **docname**  
  CHAR(128)  
  Working document or system

- **cells**  
  CHAR(128)  
  Working items

**Returns**

Data from the windows program; NULL if an error occurs (The error can be seen using the DDEGeterror function.)

**DDEPeek** gets values from the specified program and stores it in a variable. Each value retrieved by the function is separated by the tabulation character. The newline character is changed to the ASCII 13 character. For example:

```sql
CALL DDEPeek("EXCEL", "Document1", "R1C1;R2C2") RETURNING ret
```
DDE Communication Protocol

**DDEFinish — Closes a DDE Connection**

Syntax: `CALL DDEFinish(progname, docname)`

- **progname**: CHAR(128) | Program name
- **docname**: CHAR(128) | Working document or system

Returns: TRUE if the closing action has been made;
FALSE if an error occurs (The error can be seen using the DDEGeterror function.)

DDEFinish closes the specified DDE channel represented by its unique identifier. For example:

```
CALL DDEFinish("EXCEL", "Document1") RETURNING ret
```

**DDEFinishAll — Closes All DDE Connections**

Syntax: `DDEFinishAll()`

Returns: TRUE if all DDE channels have been closed;
FALSE if an error occurs (The error can be seen using the DDEGeterror function.)

DDEFinishAll closes all DDE connections, as well as the program sending or receiving data on the DDE channels. For example:

```
CALL DDEFinishAll() RETURNING ret
```

**DDEGeterror — Error Manager Function**

Syntax: `DDEGeterror()`

Returns: Error message for the current error, or NULL for no error

DDEGeterror retrieves the last error on the DDE channel. For example:

```
CALL DDEGeterror() RETURNING mess
```
Supported Versions of Windows Applications

The following are common applications running on Windows platforms that have been tested with Dynamic 4GL:

- Winword 2.0x, 6.0x, 7.x
- Excel 4.0x 5.0x, 7.x
- Access 2.0 up to 97
- Netscape Navigator 3.0

Extension of DISPLAY ARRAY Command

- BEFORE ROW statements
- BEFORE DISPLAY statements
- AFTER ROW statements
- AFTER DISPLAY statements

These statements can be used exactly as in an INPUT ARRAY.

You can use also CONTINUE DISPLAY or EXIT DISPLAY.

**Important:** The trigger BEFORE ROW is executed before BEFORE DISPLAY, whereas AFTER ROW is executed before AFTER DISPLAY.
Other Non-Graphical Extensions

Here is an example of the DISPLAY ARRAY command:

```
.
.
.
LET initdsp=TRUE
LET array_line=10
LET screen_line=5
DISPLAY ARRAY a TO scr.*
BEFORE DISPLAY
   DISPLAY "before display"
BEFORE ROW
   IF initdsp THEN
      CALL dialog.setcurrline(screen_line,array_line)
   END IF
   LET initdsp=FALSE
AFTER ROW
   LET i=arr_curr()
   DISPLAY i TO a_field
ON KEY(F22)
   LET i=arr_curr()
   IF i == 40 THEN
      EXIT DISPLAY
   END IF
AFTER DISPLAY
   DISPLAY "after display"
   LET i=arr_curr()
   IF i > 50 THEN
      CONTINUE DISPLAY
   END IF
END DISPLAY
.
.
.
```

Other Non-Graphical Extensions

This section describes the fgl_getkey(), fgl_system(), and WinExec() functions.
**Other Non-Graphical Extensions**

---

**fgl_getkey()**

This function waits for a keystroke and returns the key code of a pressed key.

**Syntax**

```plaintext
fgl_getkey()
```

**Returns**

Value of the keystroke

**Example:** The following program displays a message when you press T:

```plaintext
MAIN
DEFINE key INTEGER
--#CALL fgl_init4.js()
--#LET key = fgl_getkey()
IF key = 116 THEN
--#CALL fgl_winmessage("fgl_winmessage", "You have pressed T", "info")
END IF
END MAIN
```

If you press the T key, you receive the following message.

![Figure 6-1](image-url)

This function can be used in association with the **fgl_keyval()** function of INFORMIX-4GL.

The following tables list the values returned by the **fgl_getkey** function.

<table>
<thead>
<tr>
<th>Code</th>
<th>Hex</th>
<th>Dec</th>
<th>Hex</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>61</td>
<td>97</td>
<td>41</td>
<td>65</td>
</tr>
<tr>
<td>B</td>
<td>62</td>
<td>98</td>
<td>42</td>
<td>66</td>
</tr>
<tr>
<td>C</td>
<td>63</td>
<td>99</td>
<td>43</td>
<td>67</td>
</tr>
<tr>
<td>D</td>
<td>64</td>
<td>100</td>
<td>44</td>
<td>68</td>
</tr>
<tr>
<td>E</td>
<td>65</td>
<td>101</td>
<td>45</td>
<td>69</td>
</tr>
<tr>
<td>F</td>
<td>66</td>
<td>102</td>
<td>46</td>
<td>70</td>
</tr>
<tr>
<td>G</td>
<td>67</td>
<td>103</td>
<td>47</td>
<td>71</td>
</tr>
<tr>
<td>H</td>
<td>68</td>
<td>104</td>
<td>48</td>
<td>72</td>
</tr>
<tr>
<td>I</td>
<td>69</td>
<td>105</td>
<td>49</td>
<td>73</td>
</tr>
<tr>
<td>J</td>
<td>6a</td>
<td>106</td>
<td>4a</td>
<td>74</td>
</tr>
</tbody>
</table>

(1 of 2)
Other Non-Graphical Extensions

<table>
<thead>
<tr>
<th>Code</th>
<th>Hex</th>
<th>Dec</th>
<th>Hex</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>6b</td>
<td>107</td>
<td>4b</td>
<td>75</td>
</tr>
<tr>
<td>L</td>
<td>6c</td>
<td>108</td>
<td>4c</td>
<td>76</td>
</tr>
<tr>
<td>M</td>
<td>6d</td>
<td>109</td>
<td>4d</td>
<td>77</td>
</tr>
<tr>
<td>N</td>
<td>6e</td>
<td>110</td>
<td>4e</td>
<td>78</td>
</tr>
<tr>
<td>O</td>
<td>6f</td>
<td>111</td>
<td>4f</td>
<td>79</td>
</tr>
<tr>
<td>P</td>
<td>70</td>
<td>112</td>
<td>50</td>
<td>80</td>
</tr>
<tr>
<td>Q</td>
<td>71</td>
<td>113</td>
<td>51</td>
<td>81</td>
</tr>
<tr>
<td>R</td>
<td>72</td>
<td>114</td>
<td>52</td>
<td>82</td>
</tr>
<tr>
<td>S</td>
<td>73</td>
<td>115</td>
<td>53</td>
<td>83</td>
</tr>
<tr>
<td>T</td>
<td>74</td>
<td>116</td>
<td>54</td>
<td>84</td>
</tr>
<tr>
<td>U</td>
<td>75</td>
<td>117</td>
<td>55</td>
<td>85</td>
</tr>
<tr>
<td>V</td>
<td>76</td>
<td>118</td>
<td>56</td>
<td>86</td>
</tr>
<tr>
<td>W</td>
<td>77</td>
<td>119</td>
<td>57</td>
<td>87</td>
</tr>
<tr>
<td>X</td>
<td>78</td>
<td>120</td>
<td>58</td>
<td>88</td>
</tr>
<tr>
<td>Y</td>
<td>79</td>
<td>121</td>
<td>59</td>
<td>89</td>
</tr>
<tr>
<td>Z</td>
<td>7a</td>
<td>122</td>
<td>5a</td>
<td>90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Hex</th>
<th>Dec</th>
<th>Hex</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>2000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>2001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>2002</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>2003</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pg-Up</td>
<td>2006</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pg-Down</td>
<td>2005</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>3000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>3001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td>3002</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td>3003</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>3004</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F6</td>
<td>3005</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F7</td>
<td>3006</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F8</td>
<td>3007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F9</td>
<td>3008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F10</td>
<td>3009</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F11</td>
<td>3010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F12</td>
<td>3011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F13</td>
<td>3012</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F14</td>
<td>3013</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F15</td>
<td>3014</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F16</td>
<td>3015</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Other Non-Graphical Extensions

<table>
<thead>
<tr>
<th>Number</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>1</td>
<td>49</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>51</td>
</tr>
<tr>
<td>4</td>
<td>52</td>
</tr>
<tr>
<td>5</td>
<td>53</td>
</tr>
<tr>
<td>6</td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td>55</td>
</tr>
<tr>
<td>8</td>
<td>56</td>
</tr>
<tr>
<td>9</td>
<td>57</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPC</td>
<td>32</td>
</tr>
<tr>
<td>TAB</td>
<td>9</td>
</tr>
<tr>
<td>ESC</td>
<td>27</td>
</tr>
</tbody>
</table>

(1 of 3)
**Other Non-Graphical Extensions**

<table>
<thead>
<tr>
<th>Key</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTER</td>
<td>13</td>
</tr>
<tr>
<td>&lt;</td>
<td>60</td>
</tr>
<tr>
<td>&gt;</td>
<td>62</td>
</tr>
<tr>
<td>!</td>
<td>33</td>
</tr>
<tr>
<td>§</td>
<td>167</td>
</tr>
<tr>
<td>:</td>
<td>58</td>
</tr>
<tr>
<td>/</td>
<td>47</td>
</tr>
<tr>
<td>;</td>
<td>59</td>
</tr>
<tr>
<td>.</td>
<td>46</td>
</tr>
<tr>
<td>,</td>
<td>44</td>
</tr>
<tr>
<td>?</td>
<td>63</td>
</tr>
<tr>
<td>ù</td>
<td>249</td>
</tr>
<tr>
<td>%</td>
<td>37</td>
</tr>
<tr>
<td>*</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>181</td>
</tr>
<tr>
<td>$</td>
<td>36</td>
</tr>
<tr>
<td>£</td>
<td>163</td>
</tr>
<tr>
<td>º</td>
<td>164</td>
</tr>
<tr>
<td>Ins</td>
<td>0</td>
</tr>
<tr>
<td>Deb</td>
<td>1</td>
</tr>
<tr>
<td>Del</td>
<td>4003</td>
</tr>
<tr>
<td>Code</td>
<td>Dec</td>
</tr>
<tr>
<td>End</td>
<td>0</td>
</tr>
<tr>
<td>&amp;</td>
<td>38</td>
</tr>
<tr>
<td>é</td>
<td>233</td>
</tr>
<tr>
<td>'</td>
<td>39</td>
</tr>
<tr>
<td>(</td>
<td>40</td>
</tr>
<tr>
<td>-</td>
<td>45</td>
</tr>
<tr>
<td>è</td>
<td>232</td>
</tr>
<tr>
<td>_</td>
<td>95</td>
</tr>
<tr>
<td>ç</td>
<td>231</td>
</tr>
<tr>
<td>à</td>
<td>224</td>
</tr>
<tr>
<td>)</td>
<td>41</td>
</tr>
<tr>
<td>=</td>
<td>61</td>
</tr>
<tr>
<td>-</td>
<td>176</td>
</tr>
<tr>
<td>+</td>
<td>43</td>
</tr>
</tbody>
</table>

(2 of 3)
Other Non-Graphical Extensions

uiInkey() - Getkey in C Functions

In C functions, the equivalent of the function fgl_getkey() used to wait for a keystroke is uiInkey().

Syntax       uiInkey()

Returns      Value of the keystroke

fgl_system()

This function allows a RUN of a program needing a UNIX terminal emulator on the Windows client, even if the running F4GL program has been started without a visible terminal. The UNIX terminal will be raised and activated and then lowered later, when the program that needs it finishes.

Syntax       fgl_system(command)

In this syntax, command is a string or variable containing the commands to be run.

Running this function correctly requires the termcap entries hp (for raising the terminal) and rp (for lowering the terminal). For the Windows front end terminal emulation, the entries should have the values:

:hp=\E[0y:rp=\E[1y: \n
(3 of 3)
Other Non-Graphical Extensions

**WinExec() and WinExecWait()**

This function starts a Windows program on the machine running the Windows front end.

**Syntax**

```
WinExec(progname)
```

Starts a program on the Windows client without waiting for its end to resume execution of the 4GL program.

```
WinExecWait(progname)
```

Starts a program on the Windows client and waits for its end to resume execution.

**Example:**

```
LET var = WinExec("C:\EXCEL\EXCEL.EXE")
```

This line starts excel.exe on the Windows PC running the front end.

Those functions return TRUE if the application is successfully started. If FALSE is returned, you can see the error using the function DDEGeterror.

Four back slashes are needed as escape characters to transmit one to the client machine.
New Form Specification Fields

This section describes the 4GL extensions for the following controls:

- List boxes
- Command buttons
- Bitmaps
- Check boxes
- Radio buttons
- Combo boxes
- Scrolling fields

List Boxes

Screen arrays in graphical mode are displayed with the list box objects.

The following code generates output in graphical mode:

```plaintext
DATABASE formonly
SCREEN {
    [f01    ]   [f02    ]
    [f01    ]   [f02    ]
    [f01    ]   [f02    ]
    [f01    ]   [f02    ]
    [f01    ]   [f02    ]
    [f01    ]   [f02    ]
    [f01    ]   [f02    ]
    [f01    ]   [f02    ]
    [f01    ]   [f02    ]
    [f02    ]   [f02    ]
}
ATTRIBUTES
f01 = formonly.f01 type char;
f02 = formonly.f02 type char;
INSTRUCTIONS
SCREEN RECORD s_rec[10] (f01,f02)
```
List Boxes

If you want the fields to appear on individual lines, add the following string in the attribute section of your fields in the form specification file:

```
options="-nolist"
```

In the previous example, if you change the lines:

```
f01 = formonly.f01 type char;
f02 = formonly.f02 type char;
```

to match these:

```
f01 = formonly.f01 type char. options="-nolist";
f02 = formonly.f02 type char. options="-nolist";
```

you get the following result:
Buttons

This feature may be useful to keep the alignment of fields on forms. The list box display type does not allow you to configure the colors of the object. The **nolist** display type lets you control the color parameters.

**Buttons**

This section discusses extensions for menu buttons, hot key buttons, and in-form buttons.

**Menu Buttons**

The menu buttons created with the 4GL statements MENU … END MENU are displayed as rows or columns of buttons. You can access these buttons by using keyboard shortcuts, as with ASCII 4GL applications, or by clicking them.
Buttons

To choose the positioning of the button on the screen, you have to use the `Menu.style` resource in the `fglprofile` configuration file:

- `Menu.style = 0` The menu is set on the top of the application window.
- `Menu.style = 1` The menu is set on the right frame of the application.

For more information about the `fglprofile`, see Chapter 5, “Differences Between 4GL and Dynamic 4GL Compilers,” and Appendix B, “The Configuration File.”

Hot Key Buttons

Hot keys defined in COMMAND KEY or ON KEY statements are displayed in a separate frame located on the right side of the application. These buttons are automatically displayed when activated. You can then access hot keys by pressing the corresponding key, like with the ASCII version of the application, or by clicking the button with the mouse.

There are different ways to define the labels for each hot key button:

- The default value is the name of the key.
- The first method changes the label of a hot key for all applications.
- The second method changes the label for a specific form specification file.
- The third method changes the label for a specific field.
- The fourth method changes the label of a hot key by using calls to 4GL functions.

To use the first method, edit the `$FGLDIR/etc/fglprofile` configuration file. This file contains a section where you can define the label for each hot key. The name of this resource is:

```
key.key.text = label
```

where `key` is the name of the key (F1, CONTROL-V, ...) and `label` is the label to use on the button.

For example, the following entry in `fglprofile` changes the default label of the hot key F7 to the word `Zoom`:

```
key.f7.text = "Zoom"
```
For more information about the **fglprofile** file, see Chapter 5 and Appendix B.

The second method involves the new KEYS section in the form specification file (.per file). This method displays the label of a hot key button when the corresponding form is used.

Here is an example:

```
--# KEYS
--# F1 = "HELP ON MASK"
--# F2 = "ZOOM"
```

The --# pattern is optional, but if specified, guarantees the compatibility of your source code with INFORMIX-4GL.

The third method uses the KEY field attribute in a form specification file to change the label of hot key buttons when the cursor is in the corresponding field. Here is an example of the syntax to use:

```
ATTRIBUTES
    f001 = customer.customer_num,
    --# KEY F10 = "SEARCH",
    --# KEY F11 = "CLEAR",
    REVERSE;
```

In this example, when the cursor is in the field corresponding to the tag **f001** of the form, the labels of the F10 and F11 hot key buttons will be SEARCH and CLEAR.

The fourth method uses calls to 4GL functions in your 4GL source code modules. These functions are divided into two categories. The first set of functions must be used outside of a dialog statement (INPUT or CONSTRUCT) and the second one must be used when you want to change the label of a button while you are already in dialog.
Buttons

With these methods, if you set the label to the empty string "" the buttons will either disappear from the application window, or leave an empty button in the key frame, depending on a resource in \texttt{fglprofile}. An empty button does not react to mouse clicks. This behavior is defined in the \texttt{fglprofile} configuration file with the following resource:

\begin{verbatim}
gui.empty.button.invisible = 0  The button remains visible, but does not react to mouse clicks. This is the default value.
gui.empty.button.invisible = 1  The button becomes invisible and disappears.
\end{verbatim}

This feature does not influence the behavior of the application, however. Even if a hot key button does not appear, the user can execute the action defined by an ON KEY statement by pressing the corresponding key on the keyboard.

The order of appearance of the hot key in the right frame is defined by the \texttt{key."key_name".order} resource in \texttt{fglprofile}.

\textbf{In Form Buttons}

Buttons can be added to the screen section of a form. To do so, add a field tag to the screen section and add the \texttt{widget} and the \texttt{config} string in the attribute definition of the tag. The \texttt{widget} parameter must be set to \texttt{BUTTON} and the \texttt{config} parameter must be set to the name of the key sent to the application when the button is pressed. The following code creates a form with two buttons displayed at the bottom of the form.

In a 4GL module, add the following lines:

\begin{verbatim}
OPEN WINDOW w AT 2,3 WITH FORM "button" ATTRIBUTE(BORDER)
DISPLAY "Insert/Overwrite" TO bt1
DISPLAY "Zoom" TO bt2
INPUT BY NAME a,b,c
DISPLAY "" TO bt1 # erases label and deactivates the button.
DISPLAY "" TO bt2 # erases label and deactivates the button.
\end{verbatim}
Create this form specification file, button.per:

```plaintext
DATABASE formonly
SCREEN {
    Field1 (a )
    Field2 (b )
    Field3 (c )
    (bt ) (bt2 )
}
ATTRIBUTES
    a = formonly.a;
    b = formonly.b;
    c = formonly.c;
    bt1 = formonly.bt
        --# , widget="BUTTON", config="Control-a" ;
    bt2 = formonly.bt
        --# , widget="BUTTON", config="Control-a" ;
end
--#KEYS
  --"F1"=""
```

With this example, during the INPUT statement, you can click the two buttons. The first one will send the CONTROL-A key. This key toggles the insert and overwrite modes. The second button sends the F1 key.

**Bitmaps**

To add a picture to a form, create a field tag in the screen section of a form and add the `widget` and `config` string to the attribute definition of the tag. In this case, the `widget` parameter must be set to `BMP` and the `config` parameter must be set to the name of the bitmap file to be displayed and to the name of the key to send to the application when the bitmap is clicked.

The width of the field tag in the screen section of the form must be at least as wide as the name of the bitmaps that will be used, or it will not be possible to change them with the DISPLAY TO statement.
Check Boxes and Radio Buttons

Check boxes are used for making binary choices. Each check box controls a single variable. Check boxes in a group are not mutually exclusive options.

Radio buttons provide a way to select one of several mutually exclusive options. Several radio buttons work together to control a single variable.

In ASCII mode (with the FGLGUI environment variable set to 0), the radio buttons and check boxes will be displayed as standard Informix fields.

Check Box Syntax

In form specification files, check boxes are defined in the same manner as plain fields. But the attribute definition of the field has more options. In the following example, two check boxes will be displayed.

In the file check.per:

```informix
DATABASE FORMONLY
SCREEN {
    Check box 1:                      CheckBox 2:
    [chk01            ]               [chk02               ]
    }

ATTRIBUTES
chk01 = formonly.chk01, default="str_on"
    --# , widget="CHECK", config="str_on str_off str_lab"
    ;
chk02 = formonly.chk02, default="No"
    --# , widget="CHECK", config="Yes No acknowledge"
    ;
end
```

As usual, the --# sequences are optional and are only designed to preserve compatibility with INFORMIX-4GL.

In the attribute section of the file, the widget option is set to CHECK if you want to use check boxes.

The config option contains three parameters. The first two are respectively the values returned by the check box when it is activated and deactivated. The third parameter is the label displayed at the right side of the check box.

The check box is set to a null string if you do not specify a default value for it.
**Check Boxes and Radio Buttons**

**Important:** The length of the string returned by an active check box must be at least as long as the one returned when it is set to be inactive or the check box will behave unpredictably.

**Radio Button Syntax**

The definition of radio buttons uses the same options as the definition of check boxes. The following example will display frame that includes three radio buttons.

In the file `radio.per`:

```plaintext
DATABASE fromonly
screen {
  radiobutton:
    [rad001             ]
}

attributes
rad001 = fromonly.rad001, default="str_one"
--# ,widget="RADIO", config="str_one lab_one str_two lab_two
str_three lab_three"
; end
```

In order to use radio buttons, you have to set the `widget` attribute to `RADIO`. The `config` option is built in the following way: The "str_one" string is returned if the first radio button of the frame is selected. The "str_two" string is the value returned for the second button. The "lab_one" string is the string used for the label of the first button, "lab_two" for the second button. In both cases, subsequent strings are returned in numerical order.

In this case, the value returned by the radio button is a null string if there is no button selected in the frame. It is possible to define a default value for the radio button group.
Check Boxes and Radio Buttons

**General Specification**

You can send a single key instead of a string when you invoke a radio button or check box. The option `class=key` must be added in the attribute section of the declaration of the button or check box in the form file. For example:

```plaintext
DATABASE FORMONLY
SCREEN
{
Key Check 1 :    Key Radio 1 :
    [f05          ]   [f08          ]
    [f06          ]

} Attributes
f05=formonly.f05, class="key", widget="CHECK", config="F1 F6 {Check #1}";
f06=formonly.f06, class="key", widget="CHECK", config="F2 F7 {Check #2}";
f08=formonly.f08, class="key", widget="RADIO", config="F11 F12 F13 {Radio #1} {Radio #2} {Radio #3}";

In this example, the field f05 will send key F1 when activated and F6 when deactivated. The three choices of the radio button f08 will send F11, F12, or F13.

You can also activate or deactivate radio buttons and check boxes, but only the one from the `key` class, in 4GL programs. In order to activate a check box or radio button, use the following statement (replacing `myButton` with the name of a check box or a radio button in the current form):

```
DISPLAY "!" TO myButton
```

And to deactivate it use:

```
DISPLAY "+" TO myButton
```

If you activate a default class radio button or check box type outside of an input statement, it will appear checked but you will not be able to use it.
**Combo Boxes**

The `bmp` field is the first generation of the `combo` field object. It is an association between a `classical` field and a bitmap on its left side. It is possible to give a value to the field or to click the bitmap to send a specified key. The `bmp` fields do not require any changes to the 4GL source code to be added.

The field definition has two more attribute parameters: `widget` and `config`.

The `widget` parameter should be set to `FIELD_BMP` to indicate the type of field.

The `config` string is the name of the bitmap file with a `.bmp` extension and the name of the key sent to the application when the bitmap is clicked. The bitmap file must be in `$FGLDIR/bmp` or in `$FGLDIR/toolbars`. The default values are `$FGLDIR/toolbars/combo.bmp` for the bitmap file name and `F1` for the key. The size of the bitmap is constant, so a large bitmap will be truncated. For example:

```4GL
DATABASE formonly
  screen {
    bmp field: [bmf001             ]
  }
attributes
  bmf001 = formonly.bmp_field, widget="FIELD_BMP", config="combo.bmp Control-q";
end
```

The declaration of the key associated with the `bmp` field is case sensitive.

The function keys must be in uppercase:

`F1, F3 ...`

The first letter of the `CONTROL` key must be in uppercase:

`Control-p, Control-t ...`
Scrolling Fields

A field shorter than the corresponding program variable can be scrolled during input if the scroll attribute has been added to its definition in the form file. For example:

- In the .4gl file:

```plaintext
MAIN
DEFINE text CHAR(512)
OPEN WINDOW w1 AT 1,1 WITH FORM "demo1"
INPUT BY NAME text
END INPUT
CLOSE WINDOW w1
END MAIN
```

- In the .per file:

```plaintext
SCREEN
{                        
  Short entry: [f001     ]
}
ATTRIBUTES
f001 = formonly.text type char
         -#., scroll
;
END
INSTRUCTIONS
DELIMITERS " "
END
```

This would allow scrolling within the field up to the full length of the variable.
Graphical 4GL Extensions

In this section, some functions are provided as a workaround for specific customer needs. If you are doing new development with Dynamic 4GL, you should not need to use these functions. They are marked with the following icon:

General Display Extensions

This section describes an initialization function and other functions relating to display environments.

Fgl_init4js()

Syntax: `Fgl_init4js()`

This function is required at the beginning of every 4GL program calling new functions from the Dynamic 4GL libraries. For example:

```--#CALL fgl_init4js()```

Fgl_fglGui()

Syntax: `fgl_fglgui()`

Returns:
- TRUE if the program is run under a GUI
- FALSE if the program is run on an ASCII terminal
General Display Extensions

This function returns the current value of the FGLGUI environment variable. For example:

```gl
MAIN
  #CALL fgl_init4js()
  IF fgl_fglgui() = 1 THEN
    CALL fgl_winmessage ("Welcome from server to WTK",
                         "nice to meet you!", "info")
  ELSE
    OPEN WINDOW w1 AT 1,1 WITH 5 ROWS, 50 COLUMNS ATTRIBUTE
                  (BORDER)
    DISPLAY "Welcome from server to ASCII " AT 2, 5
    SLEEP 3
    CLOSE WINDOW w1
  END IF
END MAIN
```

Compile this program with the Dynamic 4GL compiler and execute it. If you are in graphical mode you will see the following message box.

![Figure 6-4](image)

But in ASCII mode, for the UNIX system, you will see the following screen.
General Display Extensions

Fig. 6-5

Tip: If you execute this program with a UNIX system, be sure to put a SLEEP statement after DISPLAY so that you can see the message.

Fgl_wtkClient()

This function tells you if the graphical front end used is the Windows client.

Syntax:  fgl_wtkClient()

Returns: TRUE if displayed on a Windows client;
FALSE if displayed on an X-Windows client or ASCII terminal.
The following program tests whether you are using the graphical interface and, if so, whether you are using Windows:

```
MAIN
  IF fg1_fglgui() = 1 THEN
    IF fg1_wtkclient() = 1 THEN
      CALL fg1_winmessage("Welcome from server to WTK",
                         "Pleased to meet you!!", "stop")
    ELSE
      CALL fg1_winmessage("Welcome from server to X",
                         "Nice to meet you!!", "info")
    END IF
  ELSE
    OPEN WINDOW w1 AT 1,1 WITH 5 ROWS, 50 COLUMNS WITH ATTRIBUTE (BORDER)
    DISPLAY "Welcome from server to ASCII " AT 2, 5
    SLEEP 1
    CLOSE WINDOW w1
  END IF
END MAIN
```

After compiling and executing the program, you have the two windows as in the `fgl_fglgui` examples. If you are using an X-Windows client, you will see this window.

![Window](Image)

**Figure 6-6**

### `fgl_keysetlabel()`

This function changes the label of a hot key button appearing in the right frame of the application window to allow the user to access these shortcuts with the mouse. By default the name of the corresponding key is displayed on the hot key button. If you have the line `ON KEY (f5)` in an input statement, a button labeled `f5` will be displayed in the right frame.
It is important to note that this function only works outside of a dialog piece of code. If you want to change the label of a hot key button, you must do it before the beginning of the dialog statement, such as INPUT or CONSTRUCT. If you want to change a label during a dialog, you have to use the function `dialog.keysetlabel()` instead.

**Syntax**

```plaintext
fgl_keysetlabel (hot_key_name, new_label)
```

- `hot_key_name`: The name of the hot key to change the label
- `new_label`: The new label displayed on the hot key button

The names of the keys are case sensitive. The names of the keyboard function keys are in lowercase: f1, f2, f3, and so on. For example:

```plaintext
... CALL FGL_KEYSETLABEL ("f4", "About") INPUT BY NAME f01,f02 ON KEY (f4) CALL DISPLAY_ABOUT( ) END INPUT ...
```

The label of the "f4" hot key button displayed by the ON KEY statement will be About. So the user can click the About button displayed on the right side of the application window or press F4 to execute the DISPLAY_ABOUT function.

## Dialog Boxes

This section describes the extensions that affect dialog boxes.

### fgl_winbutton()

This function displays an interactive box in a separate window with all possible answers in a menu.

**Syntax**

```plaintext
fgl_winbutton (title, text, default, buttons, icon, danger)
```

- `title`: Title of the box
- `default`: Default button selected
Dialog Boxes

You can put anything in the definition of a button, subject to the following rules:

- If you declare only one button, you can put one word on it with up to 72 characters.
- If you declare a button with a sentence as the label, you cannot put spaces between each word. Otherwise, one button will be created for each word.
- You can declare a maximum of 7 buttons with 10 characters each per call.

`fgl_winquestion()`

This function opens a dialog box with all possible answers in a menu.

<table>
<thead>
<tr>
<th>text</th>
<th>Text of the question (\n stands for new line)</th>
</tr>
</thead>
<tbody>
<tr>
<td>buttons</td>
<td>List of values separated by the pipe character (</td>
</tr>
<tr>
<td>icon</td>
<td>Name of the icon to be used in the dialog box</td>
</tr>
</tbody>
</table>

Possible Configuration

<table>
<thead>
<tr>
<th>Info</th>
<th>Exclamation</th>
<th>Question</th>
<th>Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Info" /></td>
<td><img src="image" alt="Exclamation" /></td>
<td><img src="image" alt="Question" /></td>
<td><img src="image" alt="Stop" /></td>
</tr>
</tbody>
</table>

`danger` Number of the warning item: a skull with crossbones will appear each time the pointer enters the corresponding button (on X11 only)
Dialog Boxes

Syntax:

```
fgl_winquestion(title, text, default_value, possible_values, icon, danger)
```

- **title**: Title of the dialog box
- **text**: Text of the question (\n stands for new line)
- **default_value**: Answer on which the focus has to be positioned
- **possible_values**: List of values separated by the pipe character (|)
- **icon**: Name of the icon to be used in the dialog box
- **danger**: Number of the warning item: a skull with crossbones will appear each time the pointer enters the corresponding button (on X11 only)

Returns: Text of the chosen answer
Dialog Boxes

The following program shows you how to use the `fgl_winquestion` function:

```
MAIN
DEFINE answer CHAR(100)
--#CALL fgl_init4js()
--#LET answer = fgl_winquestion ("Title of the dialog box", "Question Text", "Yes", "Yes|No|Cancel", "question",1)
END MAIN
```

This code produces the following dialog box.

```
Title of the dialog box

Question Text

Yes  No  Cancel
```

This function replaces the typical PROMPT...FOR CHAR loop.

`fgl_winmessage()`

This function formats a message and presents it in a separate window.

Syntax  
```
fgl_winmessage (title, text, icon)
```

*title*  
Title of the message box

*text*  
Text of the message

*icon*  
Name of the icon to be used in the message box
Displays a message box, with an OK button. For example:

```plaintext
MAIN
--@CALL fgl_init4js()
--@CALL fgl_winmessage("Title of the message", "Text or variable", "info")
END MAIN
```

This code produces the following message box.

![Message Box](image)

### `fgl_winprompt()`

This function displays a dialog box with a field which accepts a value.

**Syntax**

```plaintext
fgl_winprompt (x, y, text, default, length, type)
```

- `x, y` Position of the prompt window
- `text` Text of the question
- `default` Not used currently
- `length` Length of the entry
- `type` Type of variable:
  - 0  CHAR
  - 1  SMALLINT
  - 2  INTEGER
  - 7  INTEGER

**Returns** Entered value
New Input Statement Functions

Here is an example of how to use this function:

```
MAIN
DEFINE name CHAR(10)
--#CALL fgl_init4js()
--#CALL fgl_winprompt(5, 2, "Give me your name please", "", 10, 0) returning name
--#CALL fgl_winmessage("Answer", name, "info")
END MAIN
```

New Input Statement Functions

The following set of functions must be executed only inside dialog functions, such as INPUT, INPUT ARRAY, DISPLAY ARRAY, and PROMPT statements. Using these functions outside of a dialog function may create errors at compile time or at runtime.
**dialog.keysetlabel()**

Use this function for changing labels on buttons. The label of the hot key button is updated immediately.

Syntax: \[\text{dialog.keysetlabel("key", "label")}\]

The labels of the hot key buttons can be changed by the **fglprofile** configuration file, by attributes, by the KEY section in the form specification files, or with this function. Here is the priority order of the different methods to change the labels of hot key buttons. This order is from the highest to the lowest:

1. Attributes KEY in a field of a .per file (This means that you cannot change a label with **dialog.keylabel** if the same key attributes are present for a special key.)
2. The **dialog.keysetlabel** function
3. Section KEYS in a .per file
4. **fgl_keysetlabel** function
5. General level of **fglprofile**

For more information about changing labels of hot key buttons, see “Hot Key Buttons” on page 6-26.

**fgl.Buffertouched()**

This function is called by \{AFTER FIELD \| INPUT \| CONSTRUCT\} and returns a value that indicates whether or not the last field has been modified.

Syntax: \[\text{fgl_buffertouched()}\]

Returns: TRUE if the last field has been modified
New Input Statement Functions

The following source code tests if an update must be made after an input only on the last field. If something has changed during the input, a dialog box will be displayed that asks you if you want to accept the input. If not, a message appears informing you that nothing has to be done.

```
MAIN
DEFINE answer, CHAR(100),
    inst RECORD
    c1, c2, c3, c4, c5, c6, c7, c8, c9, c10 CHAR(100)
END RECORD
--#CALL fgl_init4js()
OPEN WINDOW w1 AT 1.1 WITH FORM "demo"
LET answer = "yes"

WHILE answer != "yes"
    INPUT BY NAME inst.*
    --@ AFTER FIELD c10
    --@  IF fgl_buffertouched() THEN
    --@   LET answer = fgl_winquestion("Notice", "Do you want to accept this
    --@     row"."yes", "yes|no", "info",0)
    --@  ELSE
    --@   CALL fgl_winmessage("Notice", "Nothing to be done", "info")
    --@ END IF
    --@ END INPUT
    END WHILE
CLOSE WINDOW w1
END MAIN
```

The first screen displays the form with ***** as default value.
If you exit the input without updating the row, the previous message box appears, to inform you that the row is the same as the old one. If something has changed during the input, the following dialog box appears.
New Input Statement Functions

**dialog::fieldname()**

This function returns the name of the currently prompted field.

Syntax: `dialog::fieldname()`

Returns: Name of current field

**dialog.getbuffer()**

This function returns the value of the currently prompted field.

Syntax: `dialog.getbuffer()`

Returns: Value of current field

Example: See `dialog.setbuffer()` example

**dialog.setbuffer()**

This function sets a value in the currently prompted field.

Syntax: `dialog.setbuffer(var)`

Var: Value or variable containing the value to be set in the current field

Returns: None

**dialog.setcurrline()**

This function displays a row of the program array to be set at a given line of the screen array.

Syntax: `dialog.setcurrline(scrl, progl)`

Scrl: Line of the screen array becoming current

Prog1: Line of the program array becoming current

Returns: None
New Input Statement Functions

The following example creates a display array with two “on key” options. When you press F4, the 100th row of the program record is displayed at the fifth line of the screen array, and when you press F5, the 400th row of the program record is displayed at first line of the screen array.

```
MAIN
DEFINE a ARRAY[500] OF RECORD
   c1 CHAR(10),
   c2 CHAR(12),
   c3 char(10)
END RECORD
DEFINE i INTEGER
--#CALL fgl_init4js()
FOR i = 1 TO 500
   LET a[i].c1 = i CLIPPED
   LET a[i].c2 = "555-666-" CLIPPED, a[i].c1
   LET a[i].c3 = "Washington"
END FOR
OPEN WINDOW w1 AT 1,1 WITH FORM "demo"
CALL SET_COUNT(i)
DISPLAY ARRAY a TO scr.*
--#ON KEY(f4)
--#CALL dialog.setcurrline(5,100)
--#ON KEY(f5)
--#CALL dialog.setcurrline(1, 400)
END DISPLAY
CLOSE WINDOW w1
END MAIN
```
New Input Statement Functions

Compile this program with the following form:

```c
DATABASE formonly
SCREEN
{
    CODE   Phone Number      City
    [f001] [f002] [f003]
    [f001] [f002] [f003]
    [f001] [f002] [f003]
    [f001] [f002] [f003]
    [f001] [f002] [f003]
    [f001] [f002] [f003]
    [f001] [f002] [f003]
    [f001] [f002] [f003]
}
ATTRIBUTES
    f001 = formonly.c1, UPSHIFT;
    f002 = formonly.c2, UPSHIFT;
    f003 = formonly.c3;
END
INSTRUCTIONS
DELIMITERS "  "
SCREEN RECORD scr[7] (formonly.c1,
    formonly.c2,
    formonly.c3);
  --#keys
  --#f4 = "100 th"
  --#f5 = "400 th"
END
```

After executing the program, an array appears with four buttons on the right side of the dialog box. Accept and Interrupt are created automatically by the program due to the `fglprofile` configuration.
If you press F4 or click the 100th button, the following screen array is displayed with the fifth row being current in the screen record displaying the 100th row from the program array.
New Input Statement Functions

Now press F5 or click the 400th button. The form is displayed with the first row being current and containing the 400th row of the program array.

**fgl_getcursor()**

This function returns the position of the cursor in the currently prompted field.

**Syntax**

```
fgl_getcursor()
```

**Returns**

Position of the cursor in the field

In this example, you can type a few letters in the fields and then click the getcursor button. The position of the cursor will be displayed in the error message list.
New Input Statement Functions

The 4GL source code, demo1.4gl:

```
MAIN
DEFINE text CHAR(512)
DEFINE pos INTEGER
OPEN WINDOW w1 AT 1,1 WITH FORM "demo1"
INPUT BY NAME text
   ON KEY (f4)
      --# LET pos = fgl_getcursor()
      --# MESSAGE" current position: ", pos
   END INPUT
CLOSE WINDOW w1
END MAIN
```

And the form specification file, demo1.per:

```
SCREEN
{
   Short entry: [f001
}
ATTRIBUTES
f001 = formonly.text type char
      --#, scroll
:  
END
INSTRUCTIONS
DELIMITERS " "
   --# KEYS
   --# "f4" = "getcursor"
END
```

**dialog.setcursor()**

This function sets the cursor at a defined position in the currently prompted field.

**Syntax**

```
dialog.setcursor(pos)
```

**pos**

Position in the field where the cursor has to be positioned.

If you specify a cursor position greater than the length of the variable, the cursor will disappear.
4GL Window Management Functions

This section describes the extensions that help you manage application windowing.

**fgl_setsize()**

Use the `fgl_setsize` function to change the default size of the program window.

**Syntax**

```plaintext
fgl_setsize(nblines, nbcols)
```

**nblines**  
Integer that specifies the new number of lines.

**nbcols**  
Integer that specifies the new number of columns.

**Example:**

```plaintext
IF answer = "yes" THEN
    IF reduce_flag THEN
        --#CALL fgl_setsize(25,80) #normal size
    ELSE
        --#CALL fgl_setsize(10,50) #reduced size
        LET reduce_flag = TRUE #reduced size
    END IF
END IF
```

**fgl_settitle()**

This function allows you to set the title of a program window. The default title is the program name. This can be changed using the `fgl_settitle` function.

**Syntax**

```plaintext
fgl_settitle(mytitle)
```

**mytitle**  
String or variable with the new title
Example:

MAIN
DEFINE title CHAR(100),
    flag SMALLINT
--#CALL fgl_init4js()
--#CALL fgl_settitle("hello world")
LET flag = TRUE
WHILE flag
    PROMPT"Give the new title: " FOR title
    --#CALL fgl_settitle(title)
    IF TITLE = "#" THEN
        LET flag = FALSE
    END IF
END WHILE
END MAIN

With this example, enter the new title of the window into the field and then press ENTER. To quit this program, press the interrupt key.
**4GL Window Management Functions**

---

**formfield::getoption("option")**

This function allows you to receive information about the currently prompted field during a dialog function.

**Syntax**

```
call formfield::getoption("option") returning var
```

- **option x**  
  Returns the X position of current field in the form
- **y**  
  Returns the Y position of current field in the form
- **length**  
  Returns the length of current field in the form

- **var**  
  The variable containing the return value of the function

**Example:**

```
INPUT by name f01
BEFORE INPUT
  LET LGT = formfield::getoption("length")
  MESSAGE "No more than ",LGT," characters"
END INPUT
```

---

**window::getoption("option")**

This function returns information about the current application window.

**Syntax**

```
call window::getoption("option") returning var
```

- **option name**  
  Returns the name of the current window
- **x**  
  Returns the X position of the current window
- **y**  
  Returns the Y position of the current window
- **width**  
  Returns the width of the current window
- **height**  
  Returns the height of the current window
- **border**  
  Returns TRUE if the current window has a border; otherwise returns FALSE
- **formline**  
  Returns the form line of the current window
4GL Window Management Functions

- `menuline` Returns the menu line of the current window
- `commentline` Returns the comment line of the current window
- `messageline` Returns the message line of the current window
- `errorline` Returns the error line of the current window
- `insertkey` Returns the value of `insertkey` (value as with the `fgl_getkey` function)
- `deletekey` Returns the value of `deletekey` (value as with the `fgl_getkey` function)
- `nextkey` Returns the value of `nextkey` (value as with the `fgl_getkey` function)
- `previouskey` Returns the value of `previouskey` (value as with the `fgl_getkey` function)
- `acceptkey` Returns the value of `acceptkey` (value as with the `fgl_getkey` function)
- `helpkey` Returns the value of `helpkey` (value as with the `fgl_getkey` function)
- `abortkey` Returns the value of `abortkey` (value as with the `fgl_getkey` function)
- `inputwrap` Returns TRUE if the `inputwrap` option is on; otherwise returns FALSE
- `sqlinterrupt` Returns TRUE if the `sqlinterrupt` option is on; otherwise returns FALSE
- `fieldorder` Returns TRUE if the `fieldorder` option is constraint; otherwise returns FALSE

`var` The variable containing the return value of the function
4GL Window Management Functions

Example:

```
MAIN
DEFINE VAR CHAR(20)
CALL fgl_init4js()
OPEN WINDOW hello AT 2,2 WITH 20 ROWS, 50 COLUMNS
ATTRIBUTES(BORDER)
LET var = window::getoption("name")
DISPLAY "You are in window ", var AT 5,5
SLEEP 3
CLOSE WINDOW hello
END MAIN
```

`window::current("name")`

This function makes the specified window, named `name`, the active window.

Syntax: `window::current("name")`

- `name`: Specifies the name of a window

Example:

```
Call window::current("hello")
```

`window::close("name")`

This function closes the window named `name`.

Syntax: `window::close("name")`

- `name`: Specifies the name of a window

Example:

```
CALL window::close("name")
```
Extensions for Drawing in 4GL

A set of new functions allows you to draw simple shapes. You will be able to insert lines, rectangles, ovals, circles, texts, arcs, and polygons in a defined area. You will also be able to bind a keystroke with the right or left click of the mouse on any of the items in the drawing.

Defining the Drawing Area

The drawing area is defined in the same way as a screen array. Here is an example.

In the file draw2.per:

```plaintext
DATABASE FORMONLY
SCREEN {

Enter the percentage of blue.
The rest will be filled with green.
BLUE [f01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
[c01 ]
}

ATTRIBUTES
f01 = formonly.blue;
c01 = formonly.draw.widget="Canvas";
```

The only difference is in the attributes section of the form. You must add the option `widget=Canvas` (the "Canvas" string is case sensitive).
Extensions for Drawing Shapes

Here is the complete list of all the drawing functions that can be called from your 4GL applications.

Configuration Functions

The drawinit() function is the initialization function.

Syntax: drawInit()
Returns: None

To use drawings in a 4GL program, insert the following line at the beginning of your program, before the first display open statement:

    CALL drawinit()

This function loads the graphical add-on to your client machine. If you call this function after you open the form containing the canvas, you will encounter the following problem. The calls of the canvas functions will produce no results the first time that you run your application after starting the client front-end daemon.

The DrawSelect() function selects an area in which to draw.

Syntax: DrawSelect(field_tag)

field_tag: CHAR(xx) Field tag in which you want to draw
Returns: None

Once a window containing a form with one or more drawing areas is opened, select the area in which you want to draw. All the drawing areas have fixed resolutions of 1000 by 1000 points. The 0,0 coordinate of the area is at the lower-left corner, and the 1000,1000 coordinate is at the upper-right corner. For example:

    CALL drawselect("f001")
The **drawfillcolor()** function specifies the drawing color.

**Syntax**

```plaintext
drawfillcolor(color)
```

**color**

CHAR(xx)  Name of the color

**Returns**

None

This function sets the fill color for all drawings. This function must be set before the drawing function. The color will remain active until another color is chosen. The color name list is located in the file named `rgb.txt`, located in the `FGLDIR/desi/etc/` directory on UNIX machines and in the `desi\etc\` subdirectory of the Windows front-end installation directory. For example:

```plaintext
CALL drawfillcolor("red")
```

The **drawAnchor()** function specifies the insertion point for the text.

**Syntax**

```plaintext
drawAnchor(pos)
```

**pos**

CHAR(x)  \n Top of the text

- e  Right side

- s  Bottom side

- w  Left side

**Returns**

None

Use **drawAnchor()** to you can specify the insertion point for the text before using the function **drawtext**. For example:

```plaintext
CALL drawanchor("n")
```

The **DisableColorLines()** function defines whether the color of the line can change.

**Syntax**

```plaintext
disablecolorlines(colorLines)
```

**colorLines**

INTEGER  
0  The lines take the color defined by **drawFillColor**

1  The lines are always black

**Returns**

None
Extensions for Drawing Shapes

By default, the lines take the color defined by the DrawFillColor function.

Example:

    CALL DisableColorLines(1)

The drawlinewidth() function specifies the width of the line.

Syntax

    drawlinewidth(width)

Parameters:

- **width**: INTEGER
  - Width of the line, in pixels

Returns

    None

You can set the width of the line before using the DrawLine function. For example:

    CALL drawlinewidth(2)

The drawClear() function is the clear function.

Syntax

    DrawClear()

Returns

    None

This function clears the drawing area specified by the drawselect function. For example:

    CALL drawclear()

Drawing Functions

Use the DrawRectangle() function to draw a rectangle by specifying the lower-left corner and the length.

Syntax

    DrawRectangle(y,x,dx,dy)

Parameters:

- **y**, **x**: INTEGER
  - Coordinate of the lower-left corner
- **dx**, **dy**: INTEGER
  - Length of the rectangle

Returns

    The item number of the rectangle in the canvas
The rectangle is filled with the color set using the function `drawfillcolor`. For example:

```plaintext
CALL drawrectangle(500,400,120,110) RETURNING ret
```

Use the `drawoval()` function draw an oval.

**Syntax**

```plaintext
drawoval(y,x,dy,dx)
```

- `y,x` INTEGER  Coordinate of the lower-left corner
- `dx, dy` INTEGER  Length of the rectangle that contains the oval

**Returns**  The item number of the oval in the canvas

This function draws an oval in a bounding rectangle. The rectangle is defined in the same way as with the `drawrectangle` function. The oval is filled with the color set using the function `drawfillcolor`. For example:

```plaintext
CALL drawoval(500,400,150,100) RETURNING ret
```

Use the `DrawCircle()` function to draw a circle.

**Syntax**

```plaintext
DrawCircle(y,x,r)
```

- `y,x` INTEGER  The lower-left corner of the bounding square that contains the circle
- `r` INTEGER  The border length (equivalently, the diameter)

**Returns**  The item number of the circle in the canvas

This function draws a circle in a bounding square, specifying the lower-left corner of the square and the border length. The circle is filled with the color set using the function `drawfillcolor`. For example:

```plaintext
CALL drawcircle(500,400,65) RETURNING ret
```

Use the `drawline()` function to draw a line.

**Syntax**

```plaintext
drawline(y,x,dy,dx)
```
Extensions for Drawing Shapes

- **DrawLine(y,x,dy,dx)**
  
  Parameters:
  
  - `y,x` (INTEGER): Coordinate of the first point of the line.
  - `dy,dx` (INTEGER): Coordinate of the last point of the line.

  Returns: The item number of the line in the canvas.

  This function draws a line from start point to end point using the `drawlinewidth` function. The line is filled with the color set using the function `drawfillcolor`. For example:

  ```
  CALL drawline(500,400,600,500) RETURNING ret
  ```

  **Use the DrawText() function to draw text.**

  Syntax: `DrawText(y,x,t)`

  Parameters:
  
  - `y,x` (INTEGER): The starting point of the text.
  - `t` (CHAR(xx)): The string to draw from the starting point.

  Returns: The item number of the text in the canvas.

  This function draws the specified string at the specified coordinate. Use the `drawanchor` function to define the insertion point of the text. For example:

  ```
  CALL drawtext(500,400, "Hello world!!!") RETURNING ret
  ```

  **Use the DrawArc() function to draw an arc.**

  Syntax: `DrawArc(y,x,d,start,arc)`

  Parameters:
  
  - `y,x` (INTEGER): The coordinate of the lower-left corner.
  - `d` (INTEGER): The border length.
  - `start` (INTEGER): The start angle.
  - `arc` (INTEGER): The span of the arc.

  Returns: The item number of the arc in the canvas.
Extensions for Drawing Shapes

This function draws an arc of a circle bounded by a square. You can specify the lower-left corner of the square, its border length, the start angle of the arc in degrees, and the span of the arc in degrees. The line is filled with the color set using the function \texttt{drawfillcolor}. For example:

\begin{verbatim}
CALL drawarc(500,400,100,12,25) RETURNING ret
\end{verbatim}

Use the \texttt{DrawPolygon()} function to draw a polygon.

\begin{verbatim}
Syntax \quad DrawPolygon(\textit{list})
\textit{list} \quad \text{CHAR(\textit{xx})} \quad \text{List of coordinates}
Returns \quad \text{The item number of the polygon in the canvas}
\end{verbatim}

This function draws a filled polygon defined by the list of points. The list must contain at least three points. To separate points, use spaces rather than commas. For example:

\begin{verbatim}
CALL DrawPolygon(\texttt{"120 150 200 150 400 430"}) RETURNING ret
\end{verbatim}

\textbf{Mouse Management Functions}

Use the \texttt{DrawButtonLeft()} function to define a key to be returned when you click with the left mouse button.

\begin{verbatim}
Syntax \quad DrawButtonLeft(\textit{noit},\textit{key})
\textit{noit} \quad \text{INTEGER} \quad \text{Item number returned by the function creating the object.}
\textit{key} \quad \text{CHAR(\textit{xx})} \quad \text{The name of the key to be returned when you click an item with the left mouse button.}
Returns \quad \text{None}
\end{verbatim}

This function defines a key to be returned when you click the specified item with the left mouse button. For example:

\begin{verbatim}
CALL drawbuttonleft(num_item,"F4")
\end{verbatim}
**Toolbars**

Use the **DrawButtonRight()** function to define a key to be returned when you click with the right mouse button.

**Syntax**

```
DrawButtonRight(noit, key)
```

- `noit`  INTEGER  Item number returned by the function creating the object
- `key`  CHAR(xx)  The name of the key to be returned when you click an item with the right mouse button

**Returns**  None

This function defines a key to be returned when you click the specified item with the right mouse button. For example:

```
CALL drawbuttonright(num_item, "Control-c")
```

Use the **drawClearButton()** function to remove all key binding on an item.

**Syntax:**

```
drawClearButton(noit)
```

- `noit`  INTEGER  Item number returned by the function creating the object

**Returns**  None

This function removes all key binding to the item specified by an item number. For example:

```
CALL drawclearbutton(num_item)
```

---

**Toolbars**

You can add a toolbar that contains icons representing hot keys to the top of the screen. A corresponding help tip appears when the mouse pointer is positioned over an icon.

To enable toolbar functionality, add the following line to the **fglprofile** file:

```
gui.toolBar.enabled = 1
```

To disable toolbar functionality, add the following line:

```
gui.toolBar.enabled = 0
```
After this line, you may have groups of lines, with each group corresponding to an icon on the toolbar:

```c
gui.toolBar.order.text = "keytext"
gui.toolBar.order.bmp = "bmpname"
gui.toolBar.order.hideButton = {0|1}
```

The following table describes the elements in this example.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>order</code></td>
<td>The position of the icon in the toolbar.</td>
</tr>
<tr>
<td><code>keytext</code></td>
<td>Text as specified in the label name, or the key value of a hot key the presence of which will activate the icon on the toolbar.</td>
</tr>
<tr>
<td><code>bmpname</code></td>
<td>Name of the bitmap file to use for the icon. The file <code>bmpname.bmp</code> must exist in directory <code>$FGLDIR/toolbar</code> when using X11 or in directory <code>$WTKDIR/bmp</code> of the Windows client when using Windows, where <code>$WTKDIR</code> is the installation directory of the 4GL server.</td>
</tr>
<tr>
<td>`gui.toolBar.order.hideButton = {0</td>
<td>1}`</td>
</tr>
</tbody>
</table>

The elements `keytext` and `bmpname` can be replaced by `fglSeparator`, in which case there will be an additional space at the position specified by `order`, allowing you to separate different icon groups.
Example of a `fglprofile` configuration file:

```plaintext
guiToolBar.enabled = 1
guiToolBar.1.text = "OK"
guiToolBar.1.bmp = "exclam"
guiToolBar.1.hideButton = 1
guiToolBar.2.text = "Interrupt"
guiToolBar.2.bmp = "stop"
guiToolBar.2.hidebutton = 1
guiToolBar.3.text = "fglSeparator"
guiToolBar.3.bmp = "fglSeparator"
guiToolBar.4.text = "Help"
guiToolBar.4.bmp = "ques"
guiToolBar.4.hideButton = 1
```

This configuration file generates a toolbar with three icons. The first icon is active in dialog boxes where the accept key is active. The second icon sends an interrupt signal to the application. The third icon, separated slightly from the others, is active when help is present.
This appendix provides a complete list of all environment variables for use with Dynamic 4GL.

Some variables are only available on UNIX systems. The description section for the variable will state whether the variable is only available on UNIX.

**Dynamic 4GL Product Variables**

For each variable, this appendix provides a brief description of the variable and the possible values you can set for it, along with examples of how to set the variable on the available platforms.
**FGLGUI**

<table>
<thead>
<tr>
<th>Description</th>
<th>This variable determines if the Dynamic 4GL programs will run with a character based ASCII user interface or a graphical user interface (GUI).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>0 or not set</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Default</td>
<td>0 on UNIX</td>
</tr>
</tbody>
</table>

**Korn shell**

```
$ export FGLGUI=1
```

**C shell**

```
$ setenv FGLGUI 1
```

**Microsoft DOS**

```
C:\> set FGLGUI=1
```
<table>
<thead>
<tr>
<th><strong>FGLDBPATH</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This variable contains the paths to the schema files of the databases used, separated by colons. The compiler does not use the schema tables directly, but rather its own schema file generated by fglschema.</td>
</tr>
<tr>
<td><strong>Values</strong></td>
<td>The path to the schema file</td>
</tr>
<tr>
<td><strong>Default</strong></td>
<td>Set to the current directory</td>
</tr>
<tr>
<td><strong>Korn shell</strong></td>
<td>$ export FGLDBPATH=/schema:$FGLDBPATH</td>
</tr>
<tr>
<td><strong>C shell</strong></td>
<td>$ setenv FGLDBPATH &quot;/schema:$FGLDBPATH&quot;</td>
</tr>
<tr>
<td><strong>Microsoft DOS</strong></td>
<td>C:&gt; set FGLDBPATH=C:\schema:%FGLDBPATH%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>FGLDIR</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This variable contains the path to the installation directory. This variable is required when you use either the development package or the runtime package of Dynamic 4GL.</td>
</tr>
<tr>
<td><strong>Values</strong></td>
<td>The path to Dynamic 4GL</td>
</tr>
</tbody>
</table>
| **Default** | UNIX: /usr/fgl2c  
Windows: C:\usr\fgl2c |
| **Korn shell** | $ export FGLDIR=/usr/fgl2c |
| **C shell** | $ setenv FGLDIR "/usr/fgl2c" |
| **Microsoft DOS** | C:\> set FGLDIR=C:\usr\fgl2c |
### PATH

<table>
<thead>
<tr>
<th>Description</th>
<th>This is a system variable that contains the path to the binary programs. Add the path to the Dynamic 4GL binary program.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>The path to the binary directory</td>
</tr>
<tr>
<td>Korn shell</td>
<td><code>$ export PATH=$FGLDIR/bin:$PATH</code></td>
</tr>
<tr>
<td>C shell</td>
<td><code>$ setenv PATH $FGLDIR/bin:$PATH</code></td>
</tr>
<tr>
<td>Microsoft DOS</td>
<td><code>C:\&gt; set PATH=%FGLDIR%\bin:%PATH%</code></td>
</tr>
</tbody>
</table>

### FGLCC

<table>
<thead>
<tr>
<th>Description</th>
<th>Available only on UNIX. This variable must be set when you want to compile a new runner.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The name of the C or C++ compiler</td>
</tr>
<tr>
<td>Korn shell</td>
<td><code>$ export FGLCC=gcc</code></td>
</tr>
<tr>
<td>C shell</td>
<td><code>$ setenv FGLCC gcc</code></td>
</tr>
</tbody>
</table>
**FGLRUN**

**Description**
This variable must be set to the name of the specific p-code runner when linking p-code modules using `fgl2p -o` if those modules request C functions that have been linked to this specific runner by means of the `fglmkrun` utility.

**Value**
The name of the runner that you currently use

**Default**
`FGLRUN=fglrun`

**Korn shell**
$ export FGLRUN=fglrun

**C shell**
$ setenv FGLRUN fglrun

**Microsoft DOS**
C:\> set FGLRUN=fglrun

---

**FGLLDPATH**

**Description**
The `FGLLDPATH` variable provides the p-code runner with the correct search path for p-code object files, which are dynamically linked into an executable p-code program.

**Value**
The path to the p-code modules

**Default**
The current directory

**Korn shell**
$ export FGLLDPATH=/modules:$FGLLDPATH

**C shell**
$ setenv FGLLDPATH /modules:$FGLLDPATH

**Microsoft DOS**
C:\> set FGLLDPATH=c:\modules:%FGLLDPATH%
**FGLLIBSQL**

**Description**  
Available only on UNIX. This variable specifies the complete path to the SQL library, to link with the p-code runner or the C code programs containing the interface functions to the database server.

**Value**  
Complete path to the SQL library

**Default**  
$INFORMIXDIR/lib/libfesql.a

**Korn shell**  
$ export FGLLIBSQL=$INFORMIXDIR/lib/libfesql.a

**C shell**  
$ setenv FGLLIBSQL $INFORMIXDIR/lib/libfesql.a

**FGLLIBSYS**

**Description**  
Available only on UNIX. This variable specifies the list of system libraries and flags needed to compile a p-code runner or C code programs.

**Default**  
Depends of your host operating system

**Korn shell**  
$ export FGLLIBSYS="-lm -lsocket"

**C shell**  
$ setenv FGLLIBSYS "-lm -lsocket"

**FGLSQLDEBUG**

**Description**  
If set to 1, this variable sends to the standard output debugging information about your current SQL commands in a running 4GL program.

**Value**  
0  
disables the debugging feature

1  
enables the debugging feature

**Default**  
0
## FGLDEBUGON

**Description**  
Available only on UNIX. This variable allows you to run the X11 graphical server (fglx11d) in debug mode. Each operation is redirected to the standard output. This option is not useful for debugging 4GL applications.

**Value**  
0 or not set  
- disables the debugging feature  
1  
- enables the debugging feature

**Default**  
None

**Korn shell**  
$ export FGLDEBUGON=0

**C shell**  
$ setenv FGLDEBUGON 0

### GCC Variables

These variables are only available on UNIX.

## CC

**Description**  
Available only on UNIX. This variable is set to the name of the default compiler to use when compiling C language files.

**Value**  
The name of the compiler

**Korn shell**  
$ export CC="cc"

**C shell**  
$ setenv CC cc
### GCC

<table>
<thead>
<tr>
<th>Description</th>
<th>Available only on UNIX. This variable specifies the name of the GNU C Compiler.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The name of the GNU C compiler</td>
</tr>
</tbody>
</table>

**Korn shell**

```bash
$ export GCC=gcc
```

**C shell**

```bash
$ setenv GCC gcc
```

### GCCDIR

<table>
<thead>
<tr>
<th>Description</th>
<th>Available only on UNIX. This variable specifies the directory in which the GNU C compiler is installed. This variable is used only by Dynamic 4GL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The path of the gcc installation directory</td>
</tr>
</tbody>
</table>

**Korn shell**

```bash
$ export GCCDIR=/usr/local/gcc-2.80
```

**C shell**

```bash
$ setenv GCCDIR /usr/local/gcc-2.80
```

### GCC_EXEC_PREFIX

<table>
<thead>
<tr>
<th>Description</th>
<th>Available only on UNIX. This variable specifies the path of the installation directory of the GCC compiler.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Path to the gcc installation directory</td>
</tr>
</tbody>
</table>

**Korn shell**

```bash
$ export GCC_EXEC_PREFIX=/usr/local/gcc-2.80
```

**C shell**

```bash
$ setenv GCC_EXEC_PREFIX /usr/local/gcc-2.80
```
### Environment Variables

#### PATH

**Description**
Available only on UNIX. This variable specifies a list of the directories where the operating system looks for a needed executable file.

**Value**
Path to the binary program

**Korn shell**
```
$ export PATH=$GCCDIR/bin:$PATH
```

**C shell**
```
$ setenv PATH $GCCDIR/bin:$PATH
```

#### TCL/TK Variables

These variables are available only on UNIX.

**TCLDIR**

**Description**
Available only on UNIX. This variable is used only with the TCL/TK package included in Dynamic 4GL. This variable specifies the full path to the installation directory of the TCL/TK.

**Value**
Complete path to the TCL/TK installation directory

**Korn shell**
```
$ export TCLDIR=/usr/local
```

**C shell**
```
$ setenv TCLDIR /usr/local
```
**TK_LIBRARY**

Description: Available only on UNIX. This variable specifies the full path to the TK library subdirectory.

Value: Full path to the TK library subdirectory

Korn shell: $ export TK_LIBRARY=/usr/local/lib/tk
C shell: $ setenv TK_LIBRARY /usr/local/lib/tk

**TCL_LIBRARY**

Description: Available only on UNIX. This variable specifies the full path to the TCL library subdirectory.

Value: Full path to the TCL library subdirectory

Korn shell: $ export TCL_LIBRARY=/usr/local/lib/tcl
C shell: $ setenv TCL_LIBRARY /usr/local/lib/tcl

**PATH**

Description: Available only on UNIX. This variable specifies the list of directories where the operating system looks for a needed executable file.

Value: Path to the binary program

Korn shell: $ export PATH=$TCLDIR/bin:$PATH
C shell: $ setenv PATH $TCLDIR/bin
The Configuration File

This appendix describes all the settings available in the configuration file. For each setting, this appendix provides a description, possible values, and an example of the syntax.

General Configuration Section

This section describes the settings for the general configuration section of the configuration file.

fglrun.interface

Description  Specifies which interface configuration file the graphical daemon should use. This file must be located in the $FGLDIR/etc/ directory.

Value  Resource filename

Default  fgl2c.res

Syntax  fglrun.interface="fgl2c.res"
**fglr.run.scriptname**

**Description**  Specifies which TCL/TK script is loaded when you execute the first 4GL program after the graphical daemon is started. It will search in the `$FGLDIR/etc` directory.

- Do not change this value.

**Value**  TCL/TK script file

**Default**  `fgl2c.tcl`

**Syntax**  `fglr.run.scriptname="fgl2c.tcl"`

---

**fglr.run.defaults**

**Description**  Specifies in which directory the program-specific configuration files will be searched.

**Value**  Complete path to the specific configuration files

**Default**  `$FGLDIR/defaults`

**Syntax**  `fglr.run.defaults="$FGLDIR/defaults/"`
Runtime Configuration Section

This section describes the settings that affect runtime configuration.

General Section

This section describes the general settings for the runtime configuration section of the configuration file.

fglr.run.arrayIgnoreRangeError

Description: Ignores range control in arrays. If this variable is set to 1, if x is an array x[-1] gives no error but NULL. If this variable is set to 0, x[-1] gives error -1326.

Value: 0 or 1

Default: 0

Syntax: fglrun.arrayIgnoreRangeError=1

Recommendation: Set to 1

Dialog.fieldOrder

Description: Determines whether the intermediate event triggers (AFTER/B EFORE FIELD/ROWS) are to be executed or not when moving from one field to another using the mouse. If set to 1, the intermediate event triggers are executed. If set to 0, the intermediate event triggers are not executed.

Value: 0 or 1

Default: 1

Syntax: dialog.fieldOrder=0
**report.aggregateZero**

Description: Determines the value to be returned by report aggregate functions (avg, sum, ...) when the result is NULL.

Value:
- 0 returns NULL
- 1 returns ZERO

Default: 0

Syntax: report.aggregateZero=0

**report.rightMargin**

Description: Changes the default right margin of a report. If you plan to use lines of more than 1024 characters, you must set this value.

Value: Starting character number for the right margin

Default: 255

Syntax: report.rightMargin=255

**gui.chartable**

Description: Defines a conversion file to be used for characters under GUI. It will be searched in the $FGLDIR/etc directory. You can create a file with the mkchartab utility (see Appendix D).

Value: The path from the $FGLDIR directory to the filter file

Default: Not set

Syntax: gui.chartable="iso/ansinogr"
**fglrunch.cursor.global**

Description: With a 7.x Informix database, you can choose the scope range for cursors at runtime. By default, the scope is local to the module (as in INFORMIX-4GL 4.x).

Value: 0 for local scope
       1 for global scope

Default: 0

Syntax: `fglrunch.cursor.global=0`

**fglrunch.ix6**

Description: Commands the p-code runner (fglrunch) to act like INFORMIX-4GL 6.x. See Chapter 6, “Extensions to the 4GL Language,” for more information.

Value: 0 to react like INFORMIX-4GL 4.x
       1 to react like INFORMIX-4GL 6.x

Default: 0

Syntax: `fglrunch.ix6=0`

**fglrunch.warning.logfile**

Description: Specifies if warnings are written to an error log file.

Value: Specify 0 if you do not want warnings written to an error log file
       Specify 1 if you want warnings written to an error log file

Syntax: `fglrunch.warning.logfile=0`
Graphical Daemon Autostart

This section describes the settings that control the startup of the graphical daemon.

*fglrn.server.cmd*

Description: Specifies the command used to start the GUI daemon (fglX11d).

Value: Command to start the graphical daemon

Default: fglX11d -A for UNIX systems
            fglsserv for Windows

Syntax: fglrun.server.cmd="fglsserv"

*fglrn.server.number*

Description: Specifies the maximum number of graphical servers to autostart.

Default: 100

Syntax: fglrun.server.number=50

*fglrn.server.x*

Description: With X11 and Citrix Winframe client machines, it is possible to autostart the graphical daemon on the server when a 4GL program is executed.

If FGLSERVER is defined, values specified in it will be used first. The variable DISPLAY (or WINSTATIONNAME for Winframe) determines which number of the daemon to use.

Value: The client name and port number

Default: Not set

Syntax: fglrun.server.1="client:0, client:0.0"
Microsoft Windows Section

This section describes the settings specific to Windows configurations.

**fglrun.box.error**

**Description**
Specifies the type of error redirection to use. You can use this only with a network drive solution.

**Value**
0 to display a Windows dialog box
1 to put the error on the stderr

**Default**
0

**Syntax**
`fglrun.box.error=0`

**fglrun.cmd.winnt**

**Description**
Specifies the shell command to run for the RUN WITHOUT WAITING statement on an NT machine.

A trailing space is required after the command.

**Value**
Name of the command to execute

**Default**
cmd /c

**Syntax**
`fglrun.cmd.winnt="cmd /c "`
Microsoft Windows Section

**fglrun.cmd.win95**

**Description**  
Shell command to perform the RUN WITHOUT WAITING command on a Windows 95 machine.  
A trailing space is required after the command.

**Value**  
Name of the command to execute

**Default**  
start /m

**Syntax**  
fglrun.cmd.win95="start /m  

**fglrun.remote.envvar**

**Description**  
In Windows, specifies the name of the variable used to distinguish a remote connection from a network drive solution.  
If the runner finds this variable on NT, it will export the following two environment variables to the database:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFORMIXSERVICE</td>
<td>turbo</td>
</tr>
<tr>
<td>INFORMIXPROTOCOL</td>
<td>olsoctcp</td>
</tr>
</tbody>
</table>

**Value**  
Name of the variable to distinguish remote connection from network drive

**Default**  
REMOTEADDRESS

**Syntax**  
fglrun.remote.envvar="REMOTEADDRESS"
### fglrun.remote.envvar

**Description**  This variable must be set on the NT machine using Informix 7.2x. It contains the list of all Informix variables. On NT, those variables will be exported to the database environment, not to the process environment and not to the child processes.

**Value**  The complete list of Informix variables

**Default**  
```
"CC8BITLEVEL COLLCCHAR CONRETRY CONTIME DBANSIWARN DBDATE DBLANG DBMONEY DBNLS DBPATH DBTEMP DBTIME DELIMITIDENT ESQLMF FET_BUFF_SIZE GL_DATE GL_DATETIME INFORMIXDIR INFORMIXSERVER INFORMIXSQLHOSTS LANG LC_COLLATE LC_CTYPE LC_MONETARY LC_NUMERIC LC_TIME DBALSBC DBAPICODE DBASCIIBC DBCENTURY DBCODESET DBCONNECT DBCSCONV DBCSOVERRIDE DBCSWIDTH DBFLTMSK DBMONEYSOURCE DBSS2 DBSS3"
```

### nt.withoutoob

**Description**  Determines if your NT server uses the OOB (out of band) mechanism to simulate the interrupt signal over the network.

**Value**  
- 0  use OOB
- 1  do not use OOB

**Default**  0

**Syntax**  `nt.withoutoob=0`
Microsoft Windows Section

**fglr.run.setenv.x**

**Description**
Sets an environment variable to a specific value even if the variable is already defined. For each environment variable, increment the value of $x$ by 1, to create distinct resource names.

**Value**
Name and value of the environment variable to set

**Default**
None

**Syntax**
fglr.run.setenv.0="INFORMIXDIR=c:\informix"

**fglr.run.defaultenv.x**

**Description**
Specifies the default value of an environment variable. If a variable is not found in the environment, this value will be exported. You need to define at least `INFORMIXDIR`, `INFORMIXSQLHOSTS`, `INFORMIXSERVER`, and `INFORMIXHOST` (name of machine on which the Informix database server runs) to use a remote session on NT. Increment the value of $x$ by 1 to create distinct resource names.

**Value**
Name and value of the environment variable to set

**Default**
None

**Syntax**
fglr.run.defaultenv.0="INFORMIXSQLHOSTS=\IXSERVER"
UNIX Section

This section describes the settings specific to UNIX configurations.

fglr.run.signalOOB

Description  To send the interrupt signal to the server from the client, an OOB signal is sent over the network. On some UNIX systems, the number of the OOB signal may be different from the default used by Dynamic 4GL. In this case, you can use this resource to test the signal number and then, when identified, to specify it.

Value

- 0    receive the default signal when an OOB signal is sent to the program
- -1   test the signal received when an OOB signal is sent to the program
- >0   receive a value when an OOB signal is sent to the program

Default  0

Syntax  fglrun.signalOOB=0
License Configuration Section

This section describes the settings that affect licensing.

General Section

This section describes the general settings for the license configuration section of the configuration file.

fgllic.server

Description: Name of the machine that runs the license service program. You must set this value to use the license server.

Value: Name of the license server

Default: None

Syntax: fgllic.server="ixserver"

fgllic.service

Description: Service port number to use for communication between the client and the license server.

Value: Port number

Default: 6399

Syntax: fgllic.service="7000"
**UNIX Section**

This section describes settings that are specific to licensing on UNIX systems.

### fgllic.local

**Description** Type of management of license data.

- **Value**
  - 0 if all data will be managed by the license server
  - 1 if all data will be managed by the client

- **Default** 0

- **Syntax** `fgllic.local=0`

### fgllic.ping

**Description** Time limit for the ping to detect the license server machine by a client. If you use a distant network (by RTC or ISDN), you must increase this value.

- **Value** Time unit in milliseconds

- **Default** 3000

- **Syntax** `fgllic.ping=5000`

### UNIX Section

This section describes settings that are specific to licensing on UNIX systems.

### fgllic.check

**Description** Time period between two controls of the active user list.

- **Value** Time period between check

- **Default** Value stored in `$FGLDIR/lock/data/fglcheck`

- **Syntax** `fgllic.check="10"`
GUI Section

fgllic.ps

Description  Command giving the complete process list for a machine.
Value  Command name and flag for listing all processes running on a machine
Default  ps -ae
Syntax  fgllic.ps="ps -ae"

GUI Section

This section describes the settings that affect configuration of the graphical user interface.

General GUI Section

This section describes general GUI configuration settings.

gui.button.width

Description  Specifies the size, in characters, of the buttons located in the right key button frame.
Value  Number indicating the button width, in characters
Default  15
Syntax  gui.button.width = 20
**gui.useOOB.interrupt**

Description: Enables or disables the OOB signal mechanism. If the TCP/IP stack of the client machine (especially Windows machines) does not support the OOB mechanism, you must disable it. In this case, a second method, slightly more time-consuming, is used.

Value:
- 0: disables the OOB signal on the TCP stack
- 1: enables the OOB signal on the TCP stack

Default: 1

Syntax: `gui.useOOB.interrupt = 1`

**Sleep.minTime**

Description: Specifies the minimum time (in seconds) before the interrupt button appears when you use the SLEEP statement.

Value: Number of seconds

Default: 3

Syntax: `Sleep.minTime = 5`

**gui.key.radiocheck.invokeexit**

Description: Specifies the name of a key that, if pressed when the focus is on a radio button or a check box, invokes the currently selected control and then immediately goes to the next field. It can also be set to empty string (""").

Value: Key name

Default: "Return"

Syntax: `gui.key.radiocheck.invokeexit = "Return"`
Menu GUI Section

This section describes the menu settings in the GUI section of the configuration file.

**Menu.style**

*Description*  
Specifies the display style for the menu.

*Value*  
0  Create normal horizontal menu

1  Create a menu as a button in the right key button frame on top of the hot key buttons

*Default*  
0

*Syntax*  
Menu.style=0

**gui.menu.timer**

*Description*  
Time (in milliseconds) before the menu is disabled. Useful when you switch between windows.

*Value*  
Number of milliseconds

*Default*  
100

*Syntax*  
gui.menu.timer=100
ToolBar GUI Section

This section describes the toolbar settings in the GUI section of the configuration file.

gui.toolBar.dir

Description: Specifies the name of the subdirectory from the $FGLDIR directory for UNIX clients and from the Windows front-end installation directory for Windows machines, where the bitmap files used by the toolbar are stored.

Value: Subdirectory from $FGLDIR or WTK_DIR where the bitmap files are stored

Default: $FGLDIR/toolbars for UNIX clients

WTK_DIRECTORY\toolbars for Windows clients

Syntax: gui.toolBar.dir="$FGLDIR/mytoolbars"

gui.toolBar.enabled

Description: Enables the toolbar in your program.

Value: 0 Disables the toolbar

1 Enables the toolbar

Default: 0

Syntax: gui.toolBar.enabled = 0
**Toolbar GUI Section**

**gui.toolBar.sizeY**

Description: Specifies the height (in pixels) of the toolbar.

Value: Number of pixels

Default: 26

Syntax: `gui.toolBar.sizeY = 26`

**gui.toolBar.sizeX**

Description: Specifies the width (in pixels) of a bitmap on the toolbar.

Value: Number of pixels

Default: 27

Syntax: `gui.toolBar.sizeX = 27`

**gui.toolBar.gapX**

Description: Specifies the horizontal space (in pixels) between the left border of the screen and the first bitmap.

Value: Number of pixels

Default: 2

Syntax: `gui.toolBar.gapX=2`

**gui.toolBar.gapY**

Description: Specifies the vertical space (in pixels) between the top of the screen and the bitmaps.

Value: Number of pixels

Default: 1

Syntax: `gui.toolBar.gapY=1`
### gui.toolBar.sep

**Description**  
Specifies the size of a separator in the toolbar (pixel number = gui.toolbar.sep * gui.toolBar.sizeX).

**Value**  
Number of pixels

**Default**  
"0.3"

**Syntax**  
`gui.toolBar.sep = "0.3"`

For the `gui.toolBar.0.{bmp|comments|hideButton|key|text}` parameters, the 0 stands for the position of the icon in the toolbar. For each new toolbar icon, you should increment this value by 1 to create unique resource names.

### gui.toolBar.0.bmp

**Description**  
Name of the bitmap to be used, without file extension. The specified icon must be stored in the `gui.toolBar.dir` directory.

**Value**  
Name of the bitmap file, without file extension

**Default**  
None

**Syntax**  
`gui.toolBar.0.bmp = "quest"`

### gui.toolBar.0.comments

**Description**  
Label of the key used in the toolbar. This value appears on the active help tip.

**Value**  
String containing the comment associated with the toolbar

**Default**  
None

**Syntax**  
`gui.toolBar.0.comments = "help for this program"`
### gui.toolBar.0.hideButton

**Description**  
Indicates if the key button corresponding to the icon must disappear from the key button frame. This function does not run with the horizontal menu.

**Value**  
0  The key appears in the right frame  
1  The key does not appear  

**Default**  
0

**Syntax**  
`gui.toolBar.0.hideButton = 0`

### gui.toolBar.0.key

**Description**  
Name of the key used with this icon. This variable can be used instead of `gui.toolBar.text`.

**Value**  
Key name associated with the toolbar

**Default**  
None

**Syntax**  
`gui.toolBar.0.key = "F1"`

### gui.toolBar.0.text

**Description**  
Text associated with an icon on the toolbar. Note that for menu command text, this is only available with vertical menus.

**Value**  
Text associated with the icon

**Default**  
None

**Syntax**  
`gui.toolBar.0.text = "Help"`
Toolbar GUI Section

**gui.bubbleHelp.enabled**
Description: Enables or disables tip help.
Value:
- 0: Disables the tip
- 1: Enables it
Default: 1
Syntax: `gui.bubbleHelp.enabled = 1`

**gui.bubbleHelp.color**
Description: Specifies the background color of the help tip. You can also use the configuration manager on the client side to configure it.
Value: Name of the background color
Default: "yellow"
Syntax: `gui.bubbleHelp.color = "yellow"`

**gui.bubbleHelp.disptime**
Description: Specifies the time (in milliseconds) before the help tip appears after the mouse passes over the icon.
Value: Number of milliseconds
Default: 3000
Syntax: `gui.bubbleHelp.disptime = 3000`
Screen GUI Section

`gui.bubbleHelp.offtime`

Description: Specifies the display time (in milliseconds) of the help tip.

Value: Number of milliseconds

Default: 1000

Syntax: `gui.bubbleHelp.offtime = 1000`

Screen GUI Section

This section describes the screen layout settings in the GUI section of the configuration file.

`gui.screen.size.x`

Description: Width of the screen in characters.

Value: Number of characters

Default: 80

Syntax: `gui.screen.size.x = 100`

`gui.screen.size.y`

Description: Height of the screen in characters.

Value: Number of characters

Default: 25

Syntax: `gui.screen.size.y = 40`
### Screen GUI Section

#### gui.screen.x

<table>
<thead>
<tr>
<th>Description</th>
<th>X position of an application window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>incr \hspace{1em} An incremented position (In this case, you have to set gui.screen.incrx). \hspace{1em} center \hspace{1em} Centers the window in the screen \hspace{1em} number \hspace{1em} An absolute position, in characters</td>
</tr>
<tr>
<td>Default</td>
<td>incr</td>
</tr>
<tr>
<td>Syntax</td>
<td>gui.screen.x = &quot;incr&quot;</td>
</tr>
</tbody>
</table>

#### gui.screen.incrx

<table>
<thead>
<tr>
<th>Description</th>
<th>Specifies the increment for the display of the application windows (in number characters) on the horizontal axis.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Number of characters</td>
</tr>
<tr>
<td>Default</td>
<td>3</td>
</tr>
<tr>
<td>Syntax</td>
<td>gui.screen.incrx = 3</td>
</tr>
</tbody>
</table>

#### gui.screen.y

<table>
<thead>
<tr>
<th>Description</th>
<th>Y position of an application window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>incr \hspace{1em} An incremented position (In this case, you have to set gui.screen.incr.) \hspace{1em} center \hspace{1em} Centers the window in the screen \hspace{1em} number \hspace{1em} An absolute position, in characters</td>
</tr>
<tr>
<td>Default</td>
<td>incr</td>
</tr>
<tr>
<td>Syntax</td>
<td>gui.screen.y = &quot;incr&quot;</td>
</tr>
</tbody>
</table>
### gui.screen.incry

**Description**  Specifies the increment for the display of the application windows.

**Value**  Number of characters

**Default**  2

**Syntax**  
```
gui.screen.incry = 3  
```

---

### gui.screen.withwm

**Description**  Specifies if the window can be managed by the user.

**Value**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The main window will be ignored by the window manager. The user will not be able to manipulate the window using the normal window manager mechanisms like move and resize.</td>
</tr>
<tr>
<td>1</td>
<td>Normal mode</td>
</tr>
</tbody>
</table>

**Default**  1

**Syntax**  
```
gui.screen.withwm = 0  
```
Key GUI Section

This section describes the key code settings in the GUI section of the configuration file.

**gui.key.add_function**

Description: Specifies the offset for the code sent by SHIFT-F1. If the specified value is 12, the code sent for SHIFT-F1 is F13. If the specified value is 10, the code sent for SHIFT-F1 is F11.

<table>
<thead>
<tr>
<th>Value</th>
<th>Offset for the key code SHIFT-F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>12</td>
</tr>
</tbody>
</table>

Syntax: `gui.key.add_function=12`

**gui.key.interrupt**

Description: Specifies the name of the interrupt key.

<table>
<thead>
<tr>
<th>Value</th>
<th>Name of the interrupt key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>&quot;Control-c&quot;</td>
</tr>
</tbody>
</table>

Syntax: `gui.key.interrupt = "Delete"`

**gui.key.doubleClick.left**

Description: Specifies the key code sent to the program when the left mouse button is double-clicked.

<table>
<thead>
<tr>
<th>Value</th>
<th>Name of the key code to be sent to the program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>&quot;KEY_accept&quot;</td>
</tr>
</tbody>
</table>

Syntax: `gui.key.doubleClick.left = "F30"`
Key GUI Section

### gui.key.click.right

**Description** Specifies the key code sent to the program when the right mouse button is clicked.

**Value** Name of the key code to be sent

**Default** "F36"

**Syntax**
```
gui.key.click.right = "F20"
```

### gui.key.0.translate

**Description** Allows you to map one key to another. If a key is remapped to an empty string, this disables the key. Use the file key.tcl to test your keys.

**Value** Name of the key and the returned new value

**Default** Not set

**Syntax**
```
gui.key.0.translate = "KP_Decimal comma"
```

### key."key_name".text

**Description** The label, rather than the value, of a hot key to be displayed in the right button frame.

**Value** Text for the specific key

**Default**
- key.help.text = Help
- key.accept.text = OK
- key.interrupt.text = Interrupt
- key.delete.text = Delete
- key.insert.text = Insert
- key.return.text = Return
- key.escape.text = Escape
## Key GUI Section

The following table lists keys for specific actions.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key.help.text</td>
<td>Text for the help key</td>
</tr>
<tr>
<td>key.accept.text</td>
<td>Text for the accept key</td>
</tr>
<tr>
<td>key.interrupt.text</td>
<td>Text for the interrupt key</td>
</tr>
<tr>
<td>key.delete.text</td>
<td>Text for the delete key</td>
</tr>
<tr>
<td>key.insert.text</td>
<td>Text for the insert key</td>
</tr>
<tr>
<td>key.return.text</td>
<td>Text for the return key</td>
</tr>
<tr>
<td>key.escape.text</td>
<td>Text for the escape key</td>
</tr>
<tr>
<td>key.prevpage.text</td>
<td>Text for the previous page key</td>
</tr>
<tr>
<td>key.nextpage.text</td>
<td>Text for the next page key</td>
</tr>
</tbody>
</table>

The following table lists the function keys.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key.f1.text = F1</td>
<td></td>
</tr>
<tr>
<td>key.f2.text = F2</td>
<td></td>
</tr>
<tr>
<td>key.f3.text = F3</td>
<td></td>
</tr>
<tr>
<td>key.f4.text = F4</td>
<td></td>
</tr>
<tr>
<td>key.f5.text = F5</td>
<td></td>
</tr>
<tr>
<td>key.f6.text = F6</td>
<td></td>
</tr>
<tr>
<td>key.f7.text = F7</td>
<td></td>
</tr>
<tr>
<td>key.f8.text = F8</td>
<td></td>
</tr>
<tr>
<td>key.f9.text = F9</td>
<td></td>
</tr>
<tr>
<td>key.f10.text = F10</td>
<td></td>
</tr>
<tr>
<td>key.f11.text = F11</td>
<td></td>
</tr>
<tr>
<td>key.f12.text = F12</td>
<td></td>
</tr>
<tr>
<td>key.f13.text = F13</td>
<td></td>
</tr>
<tr>
<td>key.f14.text = F14</td>
<td></td>
</tr>
<tr>
<td>key.f15.text = F15</td>
<td></td>
</tr>
<tr>
<td>key.f16.text = F16</td>
<td></td>
</tr>
<tr>
<td>key.f17.text = F17</td>
<td></td>
</tr>
<tr>
<td>key.f18.text = F18</td>
<td></td>
</tr>
<tr>
<td>key.f19.text = F19</td>
<td></td>
</tr>
<tr>
<td>key.f20.text = F20</td>
<td></td>
</tr>
</tbody>
</table>

(1 of 2)
### Key GUI Section

The following table lists the **Control modified** keys.

<table>
<thead>
<tr>
<th>key</th>
<th>Modifiers</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>key.f21</td>
<td>F21</td>
<td></td>
</tr>
<tr>
<td>key.f22</td>
<td>F22</td>
<td></td>
</tr>
<tr>
<td>key.f23</td>
<td>F23</td>
<td></td>
</tr>
<tr>
<td>key.f24</td>
<td>F24</td>
<td></td>
</tr>
<tr>
<td>key.control-a</td>
<td>Control-a</td>
<td></td>
</tr>
<tr>
<td>key.control-b</td>
<td>Control-b</td>
<td></td>
</tr>
<tr>
<td>key.control-c</td>
<td>Control-c</td>
<td></td>
</tr>
<tr>
<td>key.control-d</td>
<td>Control-d</td>
<td></td>
</tr>
<tr>
<td>key.control-e</td>
<td>Control-e</td>
<td></td>
</tr>
<tr>
<td>key.control-f</td>
<td>Control-f</td>
<td></td>
</tr>
<tr>
<td>key.control-g</td>
<td>Control-g</td>
<td></td>
</tr>
<tr>
<td>key.control-h</td>
<td>Control-h</td>
<td></td>
</tr>
<tr>
<td>key.control-i</td>
<td>Control-i</td>
<td></td>
</tr>
<tr>
<td>key.control-j</td>
<td>Control-j</td>
<td></td>
</tr>
<tr>
<td>key.control-k</td>
<td>Control-k</td>
<td></td>
</tr>
<tr>
<td>key.control-l</td>
<td>Control-l</td>
<td></td>
</tr>
<tr>
<td>key.control-m</td>
<td>Control-m</td>
<td></td>
</tr>
<tr>
<td>key.control-n</td>
<td>Control-n</td>
<td></td>
</tr>
<tr>
<td>key.control-o</td>
<td>Control-o</td>
<td></td>
</tr>
<tr>
<td>key.control-p</td>
<td>Control-p</td>
<td></td>
</tr>
<tr>
<td>key.control-q</td>
<td>Control-q</td>
<td></td>
</tr>
<tr>
<td>key.control-r</td>
<td>Control-r</td>
<td></td>
</tr>
<tr>
<td>key.control-s</td>
<td>Control-s</td>
<td></td>
</tr>
<tr>
<td>key.control-t</td>
<td>Control-t</td>
<td></td>
</tr>
<tr>
<td>key.control-u</td>
<td>Control-u</td>
<td></td>
</tr>
<tr>
<td>key.control-v</td>
<td>Control-v</td>
<td></td>
</tr>
<tr>
<td>key.control-w</td>
<td>Control-w</td>
<td></td>
</tr>
<tr>
<td>key.control-x</td>
<td>Control-x</td>
<td></td>
</tr>
<tr>
<td>key.control-y</td>
<td>Control-y</td>
<td></td>
</tr>
<tr>
<td>key.control-z</td>
<td>Control-z</td>
<td></td>
</tr>
</tbody>
</table>
Key GUI Section

**key."key_name".order**

**Description**: Specifies an order of appearance for keys. Each key has a unique priority number. The key with the lowest priority number is displayed on the top of the right key button frame.

**Value**: Order number for the specified key name

**Default**
- help 100
- accept 101
- interrupt 102
- insert 103
- delete 104
- return 105
- f1 106
- f2 107
- . .
- . .
- . .
- f69 174
- control-a 175
- control-b 176
- . .
- . .
- . .
- control-z 200
- escape 202

**Syntax**

```
key.f1.order = 1002
```
Windows GUI Section

"action_name".defKeys

Description Specifies the list of the buttons displayed in the right key button frame of dialog boxes. Each key name must be separated by a comma.

Value List of default keys that appear in each dialog box

Default Menu.defKeys = " 
InputArray.defKeys = "accept,interrupt,insert,delete"
DisplayArray.defKeys = "accept,interrupt"
Input.defKeys = "accept,interrupt"
Construct.defKeys = "accept,interrupt"
Prompt.defKeys = "return"
Sleep.defKeys = "interrupt"
Getkey.defKeys = ""

Windows GUI Section

This section describes the settings that affect the platform-specific appearance of the user interface.

gui.mswindow.button

Description Specifies whether the buttons should look like Windows buttons or like X11 buttons.

Value 0  Use X11 style buttons
       1  Use Windows style buttons

Default 0

Syntax gui.mswindow.button=0
### gui.mswindow.scrollbar

**Description**
Specifies if the scrollbars should look like Windows scrollbars or like X11 scrollbars.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Use X11 style</td>
</tr>
<tr>
<td>1</td>
<td>Use Windows style</td>
</tr>
</tbody>
</table>

**Default**
0

**Syntax**
gui.mswindow.scrollbar=0

### gui.user.font.choice

**Description**
Restricts the end user from changing the font size of the application with the Windows front-end menu at runtime.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The user is able to change the font size</td>
</tr>
<tr>
<td>1</td>
<td>The user cannot change the font size (except by changing the local.tcl file)</td>
</tr>
</tbody>
</table>

**Default**
1

**Syntax**
gui.user.font.choice=1
This appendix contains information about how to resolve issues in the following areas:

- The interruption signal
- The p-code runner and C code compilation
- Special characters and the GLS feature
- The Windows front end
- 4GL program crashes
- X11 issues
- Windows issues

### Interruption Signal

When you press the interrupt key or the **Interrupt** button, your client machine intercepts this and send it to the server. It is not possible to send an interrupt signal over the network, so Dynamic 4GL sends an MSG_OOB (out of band) message through the connected socket, which is the real interrupt message for network operations.
**Interruption Signal**

Usually, the application server receives this signal and stops the application. Problems can occur in the following situations:

- **The client TCP/IP stack does not support the OOB message.** This is often the case with the TUN TCP/IP stack from ESKER. In this case, you must disable the OOB functionality. The compiler will then send a whole command over the network to the server machine to stop the application. Add the following line in your `fglprofile` file:
  
  ```
  gui.useOOB.interrupt=0
  ```

- **The application server TCP/IP stack does not handle OOB signals.** In this case, you must also disable the OOB mechanism and use the following setting in the `fglprofile`:
  
  ```
  gui.useOOB.interrupt=0
  ```

- **The application server uses a different code number for the OOB message.** Some systems use different signals to code the OOB message. For example, the signal number changed between SCO OPEN SERVER 5.02 and SCO OPEN SERVER 5.04. To determine the received signal that your system uses, add following line in your `fglprofile`:

  ```
  fglrun.signalOOB= -1
  ```

  Then execute a 4GL program and press the interrupt key multiple times. You will see messages similar to the following on your terminal:

  ```
  Enable trappings of signal
  Received signal is 18
  (18 is subject to change depending on systems)
  Hit your interrupt key twice:
  Received signal is xx
  Received signal is xx
  ```

  The value `xx` is returned by your operating system when an OOB message is received on a socket. You can specify this number in the `fglprofile` file with the entry:

  ```
  fglrun.signalOOB= xx
  ```
P-Code Runner and C Code Compilation

This section describes how to specify which Informix libraries to use and how to find missing system libraries.

Finding Informix 7.x Libraries

The best way to specify the list of Informix libraries to use is to set the FGLLIBSQL environment variable to that list. This list of libraries changes, depending on the version of ESQL/C and the operating system. This section describes a convenient way to find out which libraries are used if the findlib.sh script failed to find them.

If you are using ESQL/C, copy the script $INFORMIXDIR/bin/esql to an empty directory. Modify this copy in order to echo the linking command. For example:

```bash
echo $CC -I$INFORMIXDIR/incl/esql $INCLUDE $A -L $INFORMIXDIR/lib \ 
-L $INFORMIXDIR/lib/esql $SLIB $OLIB $ALIB
```

Write a small ESQL/C file. For example, t.ec:

```c
main(argc,argv)
    int argc; char *argv[];
{
}
```

Compile it using your copy of ESQL:

$ ./esql t.ec

This will give you the compile statement with all the libraries used on the standard output. For example:

```bash
c -I/usr1/informix7.11/incl/esql t.c -L 
/usr1/informix7.11/lib -L /usr1/informix7.11/lib/esql -L 
/usr1/informix7.11/lib/esql 
/usr1/informix7.11/lib/esql/libsqlshr.a -L 
/usr1/informix7.11/lib/esql/libosshr.a -L 
/usr1/informix7.11/lib/esql/libasfshr.a -L 
/usr1/informix7.11/lib/esql/libgenshr.a -L
```

Common Problems and Workarounds  C-3
Finding System Libraries

In this case, you would have to set your variable FGLLIBSQL like this:

```bash
$ export FGLLIBSQL="$INFORMIXDIR/lib/esql/libsqlshr.a
$INFORMIXDIR/lib/esql/libosshr.a
$INFORMIXDIR/lib/libasfshr.a
$INFORMIXDIR/lib/esql/libgenshr.a
$INFORMIXDIR/lib/esql/libosshr.a
$INFORMIXDIR/lib/esql/libgenshr.a
/usr/lib/libtli.a
$INFORMIXDIR/lib/libnetstubshr.a
/usr/lib/libc.a
/usr/lib/libmsaa.a
/usr/lib/libbsd.a"
```

If you do not have ESQL/C development but only INFORMIX-4GL development on your system, the way to proceed is similar to that with ESQL/C, except that you will copy and modify the script `$INFORMIXDIR/bin/c4gl` and use the copy to compile a 4GL example, but then you must remove the libraries from the list that are specific to INFORMIX-4GL.

Finding System Libraries

On some operating system implementations, the libraries might have been split. When linking, you might discover some undefined symbols. For example, if the `findlib.sh` script failed to find the required libraries, one way to find the missing libraries would be as follows.

Given a missing function named `funcname`, execute the following UNIX shell command:

```bash
$ for i in /lib/*.a /usr/lib/*.a
  do
  echo $i
  nm $i | grep funcname
  done | pg
```

If the result looks like:

```
/usr/lib/libname.a
funcname| 1640|glob | 0
```
Informix 7.2x and Special Characters

If a program aborts when you use special characters (for example, a German diaeresis or a French accent) in a CONSTRUCT, it is because Global Language Support (GLS) is active with Informix 7.2x servers.

When you create the database, you must set the environment variable DB_LOCALE. If it is not set, the database will be installed with en_US.859-1.

You can view the current configuration with the following SQL statement:

```
SELECT * FROM systables WHERE tabid IN (90, 91).
```

If the database is not created with the correct configuration, it must be unloaded with dbexport, dropped, and imported with the dbimport command and with the DB_LOCALE environment variable set to the proper value (for example, de_de.8859-1 for German).

To see which local versions are supported, run:

```
$INFORMIXDIR/bin/glfiles
```

This command will create the file lcll.txt in which you will see the supported versions.
Problems with the Windows Front End

This section describes workarounds for Windows-specific problems on the front end.

Slow rlogin and Some Characters Appear Twice

With WTK, you can have a very slow `rlogin` connection and some characters can appear twice or more for each key press. This can happen if, in the file `wtksrv.ini`, you set the variable `CHECK_PASSWORD=1` and the program does not recognize the successful login (string `LOGIN_OK`).

It is possible to trace what happened during the connection process by adding the following lines to the file `wtksrv.ini`:

```
LOGIN_DEBUG=1
```

Search the login debug terminal for the line:

```
term0:switch to normal terminal mode.
```

If this line is present, you do not have a problem. If this line is not present, you must find a string `LOGIN_OK` that is common to all systems that you will connect to.

Problems with the SCO Server

With the Windows front end, if unexpected characters appear on the screen with the terminal emulation, change the following line in the `WTKDIR\wtksrv.ini` file in the `[RLOGIN]` section:

```
sendwinsize=1
```

to

```
sendwinsize=0
```

This problem appears only on SCO and Windows NT systems.
**DOS Naming Conventions**

Do not install the Windows front end in a directory that does not conform to the 8.3 naming convention. For example, you cannot install the package in a directory named `directory~1.name`, but you can install it in a directory named `mydir`.

**Installing a New Windows Client**

If you want to install a new Windows client (not an update), you must delete the following files:

```
C:\WINDOWS\WTK.INI
C:\WINDOWS\4GLSERV1.INI
```

**Graphical Daemon on Windows Gives Link Error**

If the graphical server tells you that it cannot find a link, this is probably because the TCP/IP socket protocol is not installed on the client machine. You can check if the file `winsock.dll` is in the `Windows` directory. If not, install the TCP/IP socket support on your client machine.

**4GL Program Errors**

This section describes workarounds to use if experience errors with your 4GL program.

**Internal Data File Corrupted**

On some UNIX systems (for example, SCO), you may receive the following error message:

```
Internal data file corrupted. Cannot continue this program.
```

After that, your program crashes.
Internal Data File Corrupted

This is because the process table of UNIX systems is used to retrieve internal information. This information is stored in the $FGLDIR/lock directory. To view this table, use the UNIX command `ps -ae`.

Normally, this gives the complete list of processes. But on some operating systems (such as SCO), you see only the processes of the current user if you are not the super user.

If you receive the error message, check your UNIX documentation for a command that gives the complete list of processes, and then set the environment variable `FGLPS` to this value. For example:

```bash
FGLPS="ps -ae fx"
export FGLPS
```

If there is no command that allows a non-super user to view the whole process list on the operating system, you can use the following procedure (you need a C compiler installed on your machine):

- Log in as `root`:
  ```bash
  $ cd $FGLDIR/src
  $ cc -o psall psall.c
  $ cp psall $FGLDIR/bin
  $ cd $FGLDIR/bin
  $ chown root psall
  $ chgrp root psall
  $ chmod 0755 psall
  $ chmod a+s psall
  ```

Add to your environment file (for example: `.profile`, `envcomp`, `$FGLDIR/envf4gl`) the lines:

```bash
FGLPS="psall"
export FGLPS
```

It is also possible that the file system is full and the 4GL application cannot create the internal data files in the `$FGLDIR/lock` directory. Use the command `df` to check whether you have enough free space on the file system where Dynamic 4GL is installed.
Values of Streams on SCO Machines

Stream values must be big enough, depending on the UNIX node (using TCP, NFS, or other nodes).

To check if the stream size is large enough, log in as root, use the «crash» command and the «strstat» command. The values in the FAIL column must always be zero.

For example:

```
# crash (
  dumpfile = /dev/mem, namelist = /unix, outfile = stdout
) > strstat

ITEM CONFIG ALLOC FREE TOTAL MAX FAIL
streams 512 52 460 75 53 0
queues 1424 240 1184 172 244 0
message blocks 6258 124 6134 3673 269 0
data block totals 5007 124 4883 3103 198 0
-data block size4 512 21 491 207 29 0
-data block size16 512 3 509 428 67 0
data block size64 512 31 481 2116 40 0
-data block size128 2048 54 1994 242 57 0
-data block size256 1024 15 1009 55 17 0
-data block size512 41 0 41 5 1 0
-data block size1024 52 0 52 9 1 0
-data block size2048 50 0 50 14 1 0
-data block size4096 41 0 41 5 1 0
```

SCO Open Server 5 and GCC Compiler

SCO Open Server 5 file format is ELF 32b. The GCC compiler provided by Dynamic 4GL uses file format COFF (and produces COFF binary files). Therefore, do not install GCC on this host server; instead, use your native host C compiler, which understands the COFF file format.

If you receive the following error after compiling:

```
undefined symbol __write in $FGLDIR/lib/libgcc.a
```

use the following procedure:

- Create and edit file dummywrite.c.
- Add the following C code in the file dummy.c:

  ```c
  int __write(int fd, char *c, int l) {
    return(write(fd,c,l));
  }
  ```
On an SCO Machine, No Key Buttons Appear

- Compile the file `dummywrite.c` with your native C compiler.
- Execute the shell command `ar` and apply it to the library
  `$FGLDIR/lib/libgcc.a` as follows:

```
cd $FGLDIR/lib
rv libgcc.a dummywrite.o
```

Now you are ready to link your runner.

On an SCO Machine, No Key Buttons Appear

In `/usr/lib/lang/$LANG/*/numeric`, change the dot (.) to a comma (,).

To test, call `wish` or `tclsh`, and try `expr 3.4 + 3` or `expr 3,4 + 3`. One of them must run. Our syntax uses the dot. Here is an example:

```
LANG=german
```

in `/usr/lib/lang/german/germany/*/numeric`.

You cannot make the modification directly with an editor. You have to look at the file with `od -c numeric`. For example, if you get:

```
00000000 002 , . \0 \0 \0
00000005
```

then you must use the command:

```
$ echo -n "\002,\000\000 > numeric
```

Make sure you save the original version. You should then see the file as:

```
00000000 002 . , \0 \0 \0
00000005
```
Miscellaneous Problems Under X11

This section describes various other workarounds that you will find helpful.

Nothing Appears When Starting a Program

If you are using the X11 front end and the daemon (fglX11d) is started successfully, and you can run the wish program and get the wish window, but when starting a program, nothing appears, then the problem is that the default font from the program does not exist in the database.

The solution is to run «fglfontsel» in ASCII (FGLGUI=0) with the same username as the one that runs the daemon. Select a font with ESC, restart the daemon, and then restart the program in graphical mode. Select the correct font with fglfontsel under X11.

Numlock, X11, and the Mouse

With Numlock on, some mouse features do not run under X11. To disable this effect, run the following command:

```
$ xmodmap
```

which will display lines similar to the following:

```
shift  Shift_L(0x32), Shift_R(0x3e)
lock   Caps_Lock(0x42)
control Control_L(0x25), Control_R(0x6d)
mod1   Alt_L(0x40)
mod2   Num_Lock(0x4d)
mod3   Mode_switch(0x71)
mod4
mod5
```

If you do not see Num_Lock, your keyboard is already correctly configured. Otherwise, see which modifier (mod2 in the example) corresponds to Num_Lock, and enter:

```
$ xmodmap -e "clear mod2"
```

You can add line "clear mod2" in file $HOME/Xmodmap to be correctly configured at every start of X11.
CapsLock and Scrollbar

To enable the numeric keypad with the Num_Lock key disabled, specify the following lines in the file $HOME/.Xmodmap:

```plaintext
code 63 = KP_Multiply
code 79 = KP_7
code 80 = KP_8
code 81 = KP_9
code 82 = KP_Subtract
code 83 = KP_4
code 84 = KP_5
code 85 = KP_6
code 86 = KP_Add
code 87 = KP_1
code 88 = KP_2
code 89 = KP_3
code 90 = KP_0
code 91 = KP_Decimal
code 108 = KP_Enter
code 112 = KP_Divide
```

This forces the keypad keys to send the digit as if the Num_Lock key were active. The key code might change, depending of your keyboard layout.

CapsLock and Scrollbar

If CapsLock is on, the scrollbar does not work in GUI mode with the X11 front end.

Windows Problems and Recommendations

This section describes workarounds for problems with Windows NT and workarounds for problems that involve UNIX-to-Windows configurations.

Problems with the File fgl2c.tcl on NT

Never change or edit fgl2c.tcl on Windows NT. If you do that, \^M characters will be automatically added to the end of each line, and the WTK client will not run.
Problems Using the rcp Command

If you do not have permission to use the `rcp` command from the UNIX side to access a Windows machine having Ataman remote login services installed, then do the following:

- On Windows, display the **Advanced** page in the Ataman TCP Remote Logon Services dialog box.
- In the **Rshd** and **Rexecd** areas, leave the **List of hosts allowed to connect** field empty.
  This disables both functions because you already have them with the 4GL server, and they can cause some conflicts.

Terminal Emulation Issues

With Windows front-end terminal emulation, when you open a file with `vi` and use the **DOWN ARROW** key to move the cursor down for more than one page, the lines are often displayed on the same line on the bottom of the screen without scrolling the previous lines upward.

This is because with Windows, you have a 25-line terminal. To fix this problem, in `xterm termcap`, change the definition from `li#24` to `li#25`. On some systems, you can also export the `LINES` environment variable set to 25.

Memory Fault with ESQL 7.20.TD1 with Windows NT 4.0

Informix ESQL/C Version 7.20.TD1 with Windows NT 4.0 causes a memory fault with some Informix instructions (for example, UNLOAD). You must use ESQL 7.20.TE1 to fix this problem.

How to Start a Windows Program from the UNIX Server

From Linux:

```bash
$ rsh PC_name "winexec progname.exe"
```

From your UNIX system, use the following command:

```bash
$ rcmd PC_name "winexec progname.exe"
```
To open a file directly, enter:

```
$ rcmd PC_name "winexec \"progname.exe c:/autoexec.bat\""
```

The graphical daemon must be running when you try this command.

**emm386 on Windows 3.11**

To speed up your applications on Windows 3.11 client machines, you should not use the **emm386** memory manager.
Appendix

The Character Filter: mkchartab

It is possible to create your own character filter, which converts the key codes sent by the program to the interface and vice versa. First, you must compile the C program $FGLDIR/src/mkchartab.c. This program allows you to convert an input file containing the new key code mapping to an output file that Dynamic 4GL can use.

An example of a mapping file is $FGLDIR/src/ansinogr.map. Once compiled with the mkchartab tool, this file is the same as the current $FGLDIR/etc/iso/ansinogr.ct file. You can then use the gui.chartable entry in the configuration file.

The filter source files contain two sections, an output section and an input section. The output section contains the conversion table for the characters going to the output device. The input section contains the conversion table for the characters coming from the input device.
The following example is for the file `ansinogr.map`:

```
# Character conversion ANSI => VT100
# Input section, output section
# Syntax :
# [input|output]
# x      y
# x is replaced by y
# x and y possible values are : 'x', 0xDDDD , DDDD ( D =
# digit )
# List of mapped characters :
# A`, E`, I`, O`, U`, a`, e`, i`, o`, u`,
# A', E', I', O', U', a', e', i', o', u'

output
0x8e 0xc4
0x80 0xc7
0x90 0xc9

input
0xc4 0x8e
0xc7 0x80
0xc9 0x90
```
Appendix

Configuration Manager

This appendix describes how to set properties for GUI controls on both UNIX and Windows.

On UNIX

The configuration manager is delivered with Dynamic 4GL. The file generated is located in the home directory of the current user and is named `.fgl2crc`.

To run `confdesi`, you must be in graphical mode (`FGLGUI = 1`), and the `DISPLAY` environment variable must be set. Then enter the following command at the shell prompt:

```
$ confdesi
```

On Windows

The configuration manager is delivered with the WTK client. The file generated is located in the system directory `%windir%` (%WINDIR% for Windows 95 and Windows 3.x) and is named `locals.tcl`.

To run the configuration manager, click the Dynamic 4GL Config. Manager icon.

Regardless of whether you are running the configuration manager from UNIX or Windows, you should see the Configuration Manager window.
Using the Dynamic 4GL Configuration Manager

The program interface contains three menus: **File**, **Widget** and **Help**:

- **File**: This menu lets you manage configuration files. You can open, save, and exit program functions.
- **Widget**: This menu lets you configure the different graphical widgets.
- **Help**: This menu shows the current version of the configuration tools.

**File Menu**

The **File** menu contains four items. The **Open** item allows you to read and modify an existing configuration file. The configuration manager offers the standard filename as the default, depending on the operating system (locals.tcl on Windows and $HOME/.fgl2crc on UNIX), but this default can be overwritten.

The **Save** item allows you to save the configuration file using the default name. The new configuration will overwrite the old one.

The **Save to** item allows you to save the configuration file using the specified filename. The default depends on the operating system, but you can change it to another filename.

The **Exit** item allows you to exit from the configuration manager.

**Widget Menu**

This menu lists all graphical classes in Dynamic 4GL. Each name specifies a generic class that contains several objects to configure.
The Label Object

The **Label** object contains the configuration of the **Label** item.

- **Label.** All types of labels that are not generated like the message label
- **Message.** A label generated with a DISPLAY AT, MESSAGE, or COMMENT statement
- **Error.** A label generated with the ERROR statement
- **Line.** Specifies the configuration of the separation line

For each object except for the object label, the background, border width, and relief can be configured. The object label can only have the border width configured.

The Attributes Object

The **Attributes** object contains all attributes used in the DISPLAY {AT | TO}, INPUT, CONSTRUCT, and PROMPT statements. Each item of this object can be selected individually or all in the same operation.

You can specify a different configuration for each combination of attributes. The relief will be applied only for attribute combinations using BLINK.

The Colors Object

The **Colors** object contains the different configurations of the eight colors used by INFORMIX-4GL for the foreground (color of characters and lines) and background (color for windows, toolbars, entries).

These options allow assignment of specific colors to the standard colors used in 4GL source code (WHITE, BLACK, YELLOW, MAGENTA, RED, CYAN, GREEN, BLUE). These colors may be assigned different foreground and background colors if required.

The following rules apply in the source code:

- With no attribute, the standard configuration of the widget object will be used (for example, entry background, entry active background).
- With a color attribute, the color will be applied to the foreground.
With a color attribute and the REVERSE attribute, the color will be applied to the background.

The Button Object

The Button object contains the configuration of the different types of buttons used in Dynamic 4GL:

- **Menu button.** Button generated by a COMMAND statement within a MENU statement. For this item, foreground, background, active background, relief, border width, pad X, and pad Y can be configured.

- **Horizontal menu title.** Button containing the title set by the MENU statement. For this object, background, relief, and border width can be configured.

- **Key button.** Button generated by a COMMAND KEY or ON KEY statement within a MENU, INPUT, PROMPT, or CONSTRUCT statement. For this object, background, active background, relief, border width, width, pad X, and pad Y can be configured.

- **Key BMP.** Button generated by the widget BMP form statement. For this object, background, active background, border width, pad X, and pad Y can be configured.

- **Radio button.** Button generated by the widget RADIO in the form. For this object, background, disabled foreground, active background, relief, border width, pad X, and pad Y can be configured.

- **Check box.** Button generated by the widget CHECK in the form. For this object, disabled foreground, background, active background, relief, border width, pad X, and pad Y can be configured.

Two kinds of buttons may be used: TK buttons or Windows buttons. This choice depends on the value of the tkOrgButton entry point in the file SFGLDIR/etc/fglprofile. For more information about this configuration resource, see Appendix B, “The Configuration File.”
The Field Object

The Field object contains the different field configurations used in the screen form:

- **Field.** Basic form item configuration. For this object, background, entry active background, highlight relief, and border width can be configured.
- **Screen record.** Form item using the [DISPLAY | INPUT] ARRAY statement. For this item, background, right padding, active background, highlight background, relief, and border width can be configured.
- **Canvas.** Form item used with the canvas functions. For this object, background, relief, and border width can be configured.

The Scrollbar Object

The Scrollbar object contains the configuration for the scrollbar used with the screen record. For this object, foreground, background, and active foreground can be configured.

The Screen Object

This object contains the configuration for the different window types used in Dynamic 4GL:

- **4GL window.** General configuration for the window. For this object, background, relief, and border width can be configured.
- **Screen.** This object configures the area of the window that runs the 4GL application. For this object, background, relief, and border width can be configured.
- **Menu window.** This object configures the area of the window in which the menu is displayed. For this object, background, relief, and border width can be configured.
- **Prompt window.** This object configures the area of the window that displays PROMPT statements. For this object, background, relief, and border width can be configured.
The Help Menu

- **Keys window.** This object configures the area of the window where the buttons corresponding to the INPUT, DISPLAY, ON KEY, and COMMAND KEY statements are displayed. For this object, background, relief, and border width can be configured.

  If you specify a background color for Menu or Key window, you may not see this particular color if you do not have enough space between the buttons.

**The Help Tip Object**

The Help tip object contains the configuration for the bubble used with the toolbar. For this item, you can configure background, foreground, pad X, and pad Y.

**The Help Menu**

This menu contains an About item, which specifies the current version of Dynamic 4GL.

---

**How to Configure an Object**

To configure an item, right-click the item, select conf desi to display the configuration information for the selected item.

**Configuration Types**

This section describes the configuration types that control color choice, radio button styles, and numeric fields.

**Color Choice**

This configuration type allows a color to be chosen from a palette. A set of 20 to 40 colors is displayed; the Prev and Next buttons allow other sets to be displayed. Configuration may be stopped either with the Done button, which applies the chosen color to the selected item, or with the Cancel button. Use the Old button to display the initial color.
**The Different Configurations**

This type is used by the following properties:

- **Background**
- **Foreground**
- **Active Bg**
- **Active Fg**
- **HighlightBg**

### Radio Button Choice

This configuration type allows you to choose from among a set of values. Clicking a choice applies the chosen configuration to the displayed item. Confirm by clicking **Done**, or click **Cancel** to cancel your modifications.

This type is used by the following properties:

- **Borderwidth**
- **Relief**

### Numeric Field

For this configuration type, a numeric value must be entered. Click **Apply** to show the effect of the value entered on the item displayed. After you enter a value, click **Done** or **Cancel**.

This type is used by the **Height** property.

### The Different Configurations

This section describes the various configurations you can use.

### Color Configuration

For each color configuration, the following objects appear:

- **Color object.** Displays the different colors
- **Prev.** Allows the display of the previous colors
- **Next.** Allows the display of the next colors
The Different Configurations

- **Done.** Accepts the new color configuration
- **Cancel.** Aborts the color configuration
- **Old.** Shows the previous configuration

**Background**

Allows you to specify the color choice of the normal background of an item. This configuration is applied to all items.

**Foreground**

Allows you to specify the color choice of the normal foreground of an item. This configuration is applied to the **Scrollbar** and **Color** objects.

**Active Background**

Allows you to specify the background color choice of an item when it is active. For example, a button when the pointer is positioned on it, or a field when it is accessible for input. This configuration is applied to the **Button** and **Field** objects.

**Active Foreground**

Allows you to specify the foreground color choice of an item when it is active. For example, a scrollbar when the pointer is positioned on it. This configuration is applied to the **Scrollbar** and **Field** objects.

**Highlight Background**

Allows you to specify the background color choice of an item when it is highlighted. For example, the current line of a screen record. This configuration is applied to **Field** objects.

**Relief Configuration**

The choice of an item relief style. This configuration allows you to choose one of five relief styles: raised, sunken, flat, grooved, or ridged. This configuration is applied to **Button**, **Field**, and **Screen** objects.
The Different Configurations

Border Width Configuration

The choice of an item border width style. Relief style does not appear if you select a border with a width of zero. This configuration is applied to Label (except the Label item), Button, Field, and Window objects.

More About Relief and Border Width Attributes

For every combination of the attributes BOLD, REVERSE, UNDERLINE, and BLINK, a different relief, border width, and background color may be specified by means of the menu option Attributes and the related submenus.

The following rules apply in the source code:

- Color will apply systematically in the background for DISPLAY AT, PROMPT, ERROR, MESSAGE, INPUT, and CONSTRUCT statements.
- Relief and border width will apply systematically for DISPLAY AT and PROMPT statements.
- Relief and border width will apply only if the BLINK attribute is used for ERROR, MESSAGE, INPUT, and CONSTRUCT statements.

This can be useful for hiding specific input fields on the screen:

- With the ASCII interface, this is achieved by using the following setting in the .per file:
  `DELIMITER= "";`
- With the graphical user interface, this can be achieved by defining an attribute (BLINK for instance) with flat relief and the same color as the screen background. The following command can be used to hide the field:
  `DISPLAY TO fieldname ATTRIBUTE(BLINK)xx`

Attribute for a Specific Window

The background, relief, and color can be defined independently for all Screen objects by means of the menu option Window.
This appendix lists error messages and suggested solutions for the following kinds of errors:

- Form compilation errors
- 4GL compilation errors
- Runtime errors
- License errors

## Form Compilation Errors

-200  **fglform** Version %s (compiled: %s %s)

Description: This message appears when you type **fglform -V**.

Solution: This message is for informational purposes only.

-201  Dynamic 4GL Form Compiler: **fglform**

Usage: **fglform [-s] [-h] [-V] filename**

- -V version
- -h print this text
- -s silent (no messages)

Description: you have typed **fglform -h** or run **fglform** with a bad flag.

Solution: This message is for informational purposes only.
Form Compilation Errors

-1312 FORMS statement error number %d.
   Description: An error occurred in the form at runtime.
   Solution: Edit your source file, go to the specified line, correct the error, and compile the file again.

-1314 Program stopped at %s, line number %d.
   Description: At runtime an error occurred in the specified file at the specified line. There is no .err file generated.
   Solution: Edit your source file, go to the specified line, correct the error, and compile the file again.

-1320 A function has not returned the correct number of values expected.
   Description: A function that returns several variables has not returned the correct number of parameters.
   Solution: Check your source code and compile again.

-1500 FORM: Open form %s, bad version: %s, expecting: %s.
   Description: Your runtime package is a more recent version than the version used to compile the form specification file. This can also occur if the form file has been corrupted by a 7-bit transfer mode. For example, if you use ASCII FTP mode to transfer the files from one machine to another.
   Solution: Compile your form again with the form compiler or check if your file has been corrupted.

-2000 The form %s compilation was successful.
   Description: No problem was encountered during the form compilation process.
   Solution: This message is for informational purposes only.

-2975 The display field label tag-name has not been used.
   Description: A field tag has been declared in the screen section of the form specification file but is not defined in the attributes section.
   Solution: Check your form specification file.
-7000  **fglform**: Cannot open database dictionary %s. Run **fglschema** database.

Description: If you use references to a database in your form, to compile it, Dynamic 4GL needs the database dictionary.

Solution: Run the program **fglschema** with the name of the database as a parameter, or check the value of the variable FGLDBPATH.

---

**4GL Compilation Errors**

-4900  **fglcomp**: This syntax is not supported here.

Use [screen record name.] screen field name.

-4901  **fglcomp**: Fatal INTERNAL error: %s(%d).

Description: This error occurs when an incorrect field name is used in a BEFORE FIELD or AFTER FIELD statement.

Solution: Check your 4GL source code and recompile your application.

-6003  Dynamic 4GL Compiler Version %s (compiled: %s %s)

Description: This message appears when you type **fgl2p -V** or **fglcomp -V**.

Solution: This message is for informational purposes only.

-6004  This is Dynamic 4GL Compiler

Usage: **fglcomp [-g] [-h] filename**

  - **-V** version
  - **-h** print this text
  - **-S** dump SQL
  - **-R** ignore number return values
  - **-C** C-code — default is P-code
  - **-g** implement globals in this file (internal use only)

Description: This message is displayed when you type **fglcomp -h** or if you used **fglcomp** with an unknown flag.

Solution: This message is for informational purposes only.
4GL Compilation Errors

-6011 Demo version.
Description: This message is displayed only by the demo version.
Solution: This message is for informational purposes only.

-6020 Installation: Cannot open %s.
Description: A file is missing.
Solution: Check that the file permissions are correct for the user trying to execute an application. If the file is missing, re-install the compiler package.

-6021 This is the Dynamic 4GL Compiler demonstration program.
Description: You have installed a demo version of Dynamic 4GL. This message is displayed each time you compile.
Solution: This message is for informational purposes only.

-6023 No C-code generation with Dynamic 4GL Compiler demo program.
Description: Dynamic 4GL can compile in p-code and in C Code (only for the UNIX version). But with the demo version, C code compilation is not available.
Solution: Compile your program in p-code.

-6024 Dynamic 4GL Compiler Demonstration Version %s.
Description: This message is displayed when you type fgl2p -V or fglcomp -V on the demo version.
Solution: This message is for informational purposes only.

-6100 fglcomp: Cannot open database dictionary %s.
Run fglschema database.
Description: In your source file you used the syntax database my_base, at the top of the file, before the main section. To compile the form and source code, Dynamic 4GL needs the database dictionary.
Solution: to resolve the problem, run the program fglschema and put as a parameter the name of the database, or put the DATABASE statement in the main section just after the variable declaration and before the first call to the database.
Error Messages

4GL Compilation Errors

-6101  **fglcomp**: Cannot open globals file %s.
Description: In the source, you used **GLOBALS** but the file is not in the current directory.
Solution: Copy the globals file *filename* in the current directory, or add the complete path to the globals file *filename* in the compile command, or check the name of your globals file.

-6200  **fglcomp**: The file %s cannot be created for writing.
Description: The compiler cannot create an output file at compile time.
Solution: Check that there is no filename in the directory that has the same name as the output file, but with insufficient permission for the current user to overwrite it. Also check if the user has permission to create a file in the current directory.

-6300  **fglcomp**: The module %s does not contain function %s.
Description: The specified function is not included in the named module.
Solution: Locate in your source code the call to this function and correct the module name or the function name.

-6301  **fglcomp**: No member function %s for class %s defined.
Description: The specified member function of the named class is not defined.
Solution: Locate in your source code the call to this function and correct the class name or the function name.

-6400  **fglcomp**: Wrong number of dimensions for %s[ ].
Description: An array is called with a wrong number of dimensions in your 4GL application.
Solution: Check your 4GL source code and recompile your application.

-6500  Resource error: %s: parameter expected.
Description: An unexpected error occurred.
Solution: Contact Technical Support.
Runtime Errors

-1310 Program error at %s, line number %d.
Description: Your program generates an error at runtime because of a logical mistake.
Solution: Check your 4GL source code and recompile your application.

-1311 Date: %s Time: %s
Description: This is internal information.
Solution: No solution is required.

-1400 GUI: Cannot connect to gui.
Description: You have run a GUI application but the environment variable DISPLAY or FGLSERVER is not set correctly.
Solution: Before running the GUI application, check your environment variables. FGLSERVER must be set on the graphical server machine. This is the machine that executes the fglX11d daemon for a UNIX system or the 4GL server for Windows systems. DISPLAY must be set on the client machine. For Windows, this variable cannot be set. Also check if the graphical daemon is running.

-1401 GUI: Cannot write to gui.
Description: You have run a GUI application but the environment variable DISPLAY or FGLSERVER is not set correctly.
Solution: Before running the GUI application, check your environment variables. FGLSERVER must be set on the graphical server machine. This is the machine that executes the fglX11d daemon for a UNIX system or the 4GL server for Windows systems. DISPLAY must be set on the client machine. For Windows, this variable cannot be set. Also check if the graphical daemon is running.
-1402 GUI: Cannot read from gui.

Description: You have run a GUI application but the environment variable DISPLAY or FGLSERVER is not set correctly.

Solution: Before running the GUI application, check your environment variables. FGLSERVER must be set on the graphical server machine. This is the machine that executes the fglX11d daemon for a UNIX system or the 4GL server for Windows systems. DISPLAY must be set on the client machine. For Windows, this variable cannot be set. Also check if the graphical daemon is running.

-1403 GUI: Wrong script (fgl2c.tcl) version. Check installation.

Description: The graphical daemon has loaded a version of the front end different from the one defined in the resource files of the current version as defined by the $FGLDIR variable.

Solution: Stop and restart the graphical daemon each time you change the graphical front-end version.

-1404 GUI: Wrong server (wtkclt) version. Check installation.

Description: The graphical daemon has loaded a version of the front end different from the one defined in the resource files of the current version as defined by the $FGLDIR variable.

Solution: Stop and restart the graphical daemon each time you change the graphical front-end version.

-1406 GUI: Cannot open server file. Check installation.

Description: A file on the server side cannot be sent to the graphical interface.

Solution: Check the permission of the file located in the $FGLDIR/etc directory. These files must have at least read permission for the current user.


Description: You must set the FGLSERVER environment variables, as well as the entry of the autostart feature in $FGLDIR/etc/fglprofile.

Solution: Set the needed environment variables or add values in the $FGLDIR/etc/fglprofile file to enable the graphical daemon autostart feature.
Runtime Errors

-1408 GUI: server autostart: unknown workstation: set \texttt{fglrun.server.number = aliaslist}.

  Description: The machine described by the entry \texttt{fglrun.server.##} in the \texttt{fglprofile} file is not accessible on the network.

  Solution: Check if the machine name is correctly set in the \texttt{DISPLAY} or \texttt{FGLSERVER} variables.

-1409 GUI: Not connected. Cannot write to gui.

  Description: The communication between the 4GL application and the graphical front end is broken.

  Solution: Check if the \texttt{$FGLSERVER} and the \texttt{$DISPLAY} variables are correctly set. Also check if the daemon of the graphical front end is running.

-1410 GUI: Not connected. Cannot read to gui.

  Description: The communication between the 4GL application and the graphical front end is broken.

  Solution: Check if the \texttt{$FGLSERVER} and the \texttt{$DISPLAY} variables are correctly set. Also check if the daemon of the graphical front end is running.

-1900 INSTALLATION: Cannot open file \texttt{%s}.

  Description: The compiler cannot access the resource file \texttt{\$FGLDIR/etc/fgl2c.res}.

  Solution: Check the permissions of the resource file and change them as needed. The current user should have read permission on this file.

-1901 INSTALLATION: No such interface capability: \texttt{%s}.

  Description: The resource files from the graphical front end are from different versions. This is often caused by installing an update over an old version of the compiler. Due to permission problems, some files have been overwritten while others have not.

  Solution: Check the permissions of the files located in the \texttt{\$FGLDIR} directory and re-install the update.
-1902 INSTALLATION: %s wrong version. Expecting %s.

Description: The resource files located in the $FGLDIR/etc directory have a bad version number.

Solution: This problem often results from installing a new version of the compiler over an old one. Reinstall the new version but take care that the user doing this operation has the correct permission to overwrite the files in the $FGLDIR directory.

-1903 INSTALLATION: Cannot open factory profile %s.

Description: The $FGLDIR/etc/fglprofile file is missing or is unreadable.

Solution: Check the permission of the file. If the file is missing, reinstall the compiler.

-1904 INSTALLATION: Cannot open customer profile %s.

Description: The configuration file defined by the FGLPROFILE environment variable is missing or unreadable.

Solution: Check if the FGLPROFILE variable is correctly set and if the file is readable by the current user.

-1905 INSTALLATION: Cannot open application resources %s.

Description: The directory specified by the fglrun.default entry in $FGLDIR/etc/fglprofile is missing or not readable for the current user.

Solution: Check if the entry fglrun.default is correctly set in $FGLDIR/etc/fglprofile and if the directory specified is readable by the current user.

-1906 GUI: Cannot open char table file %s. Check your fglprofile.

Description: This error occurs if the conversion file defined by the gui.chartable entry, in the $FGLDIR/etc/fglprofile file, is not readable by the current user.

Solution: Check if the gui.chartable entry is correctly set and if the specified file is readable by the current user.
**Runtime Errors**

-1910 Internal error in the runtime library file %s.

Description: Something unpredictable occurs, generating an error.

Solution: Contact Technical Support.


Description: You used the channel extension in your program. The statement `channel::open_file` returns this error because the file that you want to open is not in the specified directory.

Solution: Check your source, and compile your source.

-2001 Unsupported mode for open file.

Description: You used the channel extension in your program. The file that you want to open does not support the specified mode.

Solution: Check the permissions for the specified file or change the `channel::open_file` statement.

-2002 Cannot open pipe.

Description: You used the channel extension in your program. The `channel::open_pipe` statement has an error because the specified command does not exist.

Solution: Check your system for the command and the source for the syntax for the command argument.

-2003 Unsupported mode for open pipe.

Description: You used the channel extension in your program. The file that you want to open does not support the specified mode.

Solution: Check the permissions for the specified file or change the `channel::open_file` statement.

-2004 Cannot write to unopened file or pipe.

Description: You used the channel extension in your program. You are trying to write data on a handle that refers to an unopened pipe.

Solution: Check your syntax.
-2005  Channel write error.
        Description: You used the channel extension in your program. You are trying to write a handle that refers to a file or pipe for which you do not have the proper syntax.
        Solution: Check your syntax.

-2006  Cannot read from unopened file or pipe.
        Description: You used the channel extension in your program. You are trying to read data from a handle that refers to an unopened pipe.
        Solution: Check your syntax.

-5000  `fglrun`: Module %s: The function %s will be called as %s.
        Description: An incorrect number of parameters are used to call a 4GL function.
        Solution: Check your source code and recompile your application.

-5001  `fglrun`: Module %s: Bad version: Recompile your sources.
        Description: You have compiled your program with an old version, the new p-code version of your program is not supported.
        Solution: Compile all source files and form files again.

-5002  `fglrun`: File %s: Bad magic number: Code cannot run with this p-code machine.
        Description: You have compiled your program with an old version, the new p-code version of your program is not supported. You might also have a file with the same name as the .42r. You used the `fglrun 42r-Name` without specifying the extension.
        Solution: To resolve this problem, call `fglrun` with the .42r extension or recompile your application.
Runtime Errors

-5003  **fglrun**: Module %s: The function %s has already been defined in module %s.
Description: The specified function is defined for the second time in the application. The second occurrence of the function is in the specified module.
Solution: Eliminate one of the two function definitions from your source code.

-5004  **fglrun**: Module %s: Unknown opcode.
Description: An unknown p-code instruction was found in the p-code application.
Solution: Check that the version of the Dynamic 4GL package executing the p-code is the same as the one that compiled the application. It is also possible that the p-code module has been corrupted. In this case you need to recompile your application.

-5005  **fglrun**: internal error: Alignment.
Description: This is an internal error, which should not normally occur.
Solution: Please contact Technical Support.

-5006  **fglrun**: The dynamic loader cannot open module %s.
Description: The module %s is not in the current directory or in one of the directories specified by the environment variable FGLLDPATH.
Solution: Set the variable FGLLDPATH.
Please see Appendix A, “Environment Variables,” for more information.

-5007  **fglrun**: The dynamic loaded module %s does not contain the function %s.
Description: A 4GL module has been changed and recompiled, but the different modules of the application have not been linked afterward.
Solution: Link the new modules together before executing your application.

-5008  **fglrun**: Module %s: already loaded.
Description: A module is loaded twice at runtime. This can occur because one module has been concatenated with another.
Solution: Recompile and relink your 4GL modules.
-5010 Internal error: type conversion not supported.

Description: In one of the 4GL source modules, a value of a type is converted into another incompatible type.

Solution: Check your source code and recompile your application.

-5100 **fglrn**: Usage: fglrun [options] program

Description: You have run the program fglrun without an argument.

Solution: This message is for informational purposes only.

-6018 Cannot access internal data file. Cannot continue this program. Please check your environment.

Description: When a client machine starts an application on the server, the application stores data in the directory $FGLDIR/lock. The client should have permission to create and delete files in this directory.

Solution: Either change the permissions of the $FGLDIR/lock directory or connect to the server with a user name that has the correct permissions.

-6019 In Dynamic 4GL Runner demonstration, only one user is available.

Description: The demo version is designed to run with only one user. Another user or another graphical daemon is currently active (4GL Server for Windows or fgIX11d for the X11 environment).

Solution: Wait until the user stops the current program, or use the same graphical daemon.

-6020 Installation: Cannot open %s.

Description: Either the file $FGLDIR/lib/fgl2c.init or the file $FGLDIR/lib/fgl.4gl cannot be read by the current user.

Solution: Check that the files exist and that they are readable for the current user.

-6021 This is Dynamic 4GL Compiler demonstration program.

Description: You are executing a program using the demo version of Dynamic 4GL.

Solution: This message is for informational purposes only.
Runtime Errors

-6022 Dynamic 4GL Runner demonstration time has expired. Please run this program again.

Description: The runtime demo version is valid only for a few minutes after you have started a program.

Solution: Restart the program.

-6023 No C-code generation with Dynamic 4GL Compiler demo program.

Description: Although Dynamic 4GL can compile in p-code and in C code (only for the UNIX version), C code compilation is not available in the demo version.

Solution: Compile your program in p-code.

-6024 Dynamic 4GL Compiler Demonstration Version %s.

Description: You have typed fglcomp -V or fgl2p -V.

Solution: This message is for informational purposes only.

-6025 Dynamic 4GL Compiler demo time has expired. Please contact your reseller.

Description: The demo version of Dynamic 4GL has a time limit of 30 days.

Solution: Either reinstall a new demo version or call your Dynamic 4GL distributor.

-6026 Bad link for Dynamic 4GL Runner demonstration. Please retry or rebuild your runner.

Description: The runner is corrupted.

Solution: Relink your runner with the fglmkrun tool.

-6102 Unknown user name. Please set environment variable USERNAME or LOGNAME.

Description: In order to start an application, the compiler should know which user is executing the program. To do so, the compiler checks one of the two variables USERNAME or LOGNAME.

Solution: Depending on your system, one of these two variables is set by the system. If not, add one of these to your environment.
License Errors

-6103 Bad format of resource %s value %s: you must use next syntax
%s=VARNAME=value.

Description: In the $FGLDIR/etc/fglprofile file, a fglrun.setenv.x or a
fglrun.defaultenv.x entry is incorrectly set.

Solution: Check your configuration file and correct the error.

-6104 Cannot put in process environment the next variable: %s

Description: A variable defined in the $FGLDIR/etc/fglprofile file by the
entry fglrun.setenv.x or fglrun.defaultenv.x, because of a system problem,
cannot be exported to the environment.

Solution: This error is caused by your system.

-6105 INFORMIXDIR environment variable is not set. Please check your
environment.

Description: The INFORMIXDIR environment variable is not set. This value
is required by the Dynamic 4GL compiler.

Solution: Set the variable to the name of the directory where the Informix
products are installed.

License Errors

-6012 Invalid serial number.

Description: You might have a different value between the FGLDIR variable
and the path to the Dynamic 4GL binary files defined in the PATH variable.

Solution: Set the environment variable FGLDIR. Then update your path using
the following commands:

Korn Shell: $ export PATH=$FGLDIR/bin:$PATH
C Shell: $ setenv PATH $FGLDIR/bin:$PATH
Microsoft DOS: C: \> set PATH=%FGLDIR%\bin;%PATH%
License Errors

-6013  Time limited version: time over.
  Description: You have installed a demo version or a time limited version and
  the valid period has expired.
  Solution: Call your Dynamic 4GL distributor to purchase Dynamic 4GL.

-6014  Your serial number is not valid for this version.
  Description: You have installed a demo version or runtime version and now
  you are using a final version or a development version.
  Solution: Call your Dynamic 4GL distributor to purchase Dynamic 4GL.

-6015  Invalid serial number answer.
  Description: It is not possible for the application to check the license validity.
  Solution: Check the permissions for all the files located in the $FGLDIR
directory. You need to have read permissions on all the files and write
permissions on the $FGLDIR/lock directory.

-6016  Problem reading Dynamic 4GL runtime license. Please check your
  environment and your license (run fglWrt -a see).
  Description: The application is unable to check the license validity.
  Solution: Check the permissions for all the files located in the $FGLDIR
directory. You must have read permissions on all the files and write permis-
sions on the $FGLDIR/lock directory. It is also possible that the FGLDIR
variable is set incorrectly but that the $FGLDIR/bin directory is set correctly
in the PATH variable.

-6017  Users limit exceeded. Cannot run this program.
  Description: There are too many users for this license. Each graphical
daemon uses one user. For example, if you have a license for 10 users, you can
start 10 graphical daemons (4GL server for Windows or fglX11d for UNIX). In
ASCII mode, each TTY running an application counts as one user.
  Solution: Wait until a user stops a graphical daemon or call your Dynamic
4GL distributor to purchase more licenses.
-6027 Cannot access the license server. Please check following things:

- Value of the variable `fglic.server`
- If the license server machine is running
- If the Dynamic 4GL License Server is running

Description: You have not specified a value for the environment variable `fglic.server` in the `$FGLDIR/etc/fglprofile` file.

Solution: Check the `fglprofile` file for the entry point `fglic.server` and specify the name (in uppercase letters) of the machine that runs the Dynamic 4GL License Server.

-6029 Unknown parameter `%s` for checking.

Description: There is a wrong parameter on the command line of the `fglWrt` tool.

Solution: Check your command line parameters, and try the command again.

-6030 With `-L` option, you must specify the serial number before the license key.

Description: You have used the `fglWrt` program with the `-L` flag but you have entered an incorrect serial number or the license key before the serial number.

Solution: Specify a valid serial number.

-6031 Expired temporary runtime license.

Description: Your temporary runtime license has expired.

Solution: Call your Dynamic 4GL distributor to get a new license.

-6032 `%s`: illegal option -- `%c`

Description: The specified program (`%s`) has been called with a specified parameter (`%c`) that is not recognized by the program.

Solution: Run the program using the flag `-h` or `--h` to get help information about it.
License Errors

-6033  %%s: option requires an argument -- %c

Description: You cannot use this option of the fglWrt tool without a parameter.

Solution: Check your command line and try the command again.

-6034  Dynamic 4GL license program

Usage: fglWrt [-l | -v | -V | -k key | -r | -s | -d | -a par] where

- l       Install a Dynamic 4GL license
- v | -V  Version number
- u       Verify current users
- k key  For runtime, deferred installation number key
- r       Run the Dynamic 4GL License Server
- s       Stop the Dynamic 4GL License Server
- d       Delete the Dynamic 4GL license
- w       Check user research
- a par  Check or view options, possible parameter values:
           who       Check user research
           view     View active users
           activ    Check active users
           see      Give current serial number

Description: This message is displayed if you have run fglWrt with the -h flag or with an invalid flag.

Solution: This message is for informational purposes only.

-6035  Error on reading directory.

Description: The compiler cannot access the $FGLDIR/lock directory. The current user must have read and write permissions in this directory.

Solution: Give the current user read and write permissions to the $FGLDIR/lock directory.
License Errors

-6041 Problem searching license information. Please call Technical Support.

Description: An error occurred during the license verification process.

Solution: Restart your program. If this does not solve the problem, check that you have installed the license by typing the command `fglWrt -a see` to read the current serial number. If you have not activated the license, run the program `fglWrt` with the flag `-L` for UNIX systems, or click License registration for Windows environments.

-6042 Bad license information. Please verify if you have installed the Dynamic 4GL license.

Description: You have attempted to run Dynamic4GL without a valid license.

Solution: Check that you have installed the license by typing the command `fglWrt -a see` to read the current serial number. If you have not activated the license, run the program `fglWrt` with the flag `-L` for UNIX systems, or click License registration for Windows environments.

-6043 Your test time is finished. You must install a new license.

Description: The test time license of Dynamic 4GL has expired.

Solution: Please call your Dynamic 4GL distributor to purchase a new license.

-6044 Bad information in license program. Please verify that you have installed the Dynamic 4GL license. For the Dynamic 4GL License Server, check that you are on the right machine.

Description: The compiler checks important software and hardware components to validate the license. If any of these components change, the license is no longer valid.

Solution: Restore the changed components or enter a new serial number.

-6045 Bad information in license program. Please verify that you have installed the Dynamic 4GL license, or for the Dynamic 4GL License Server, check that you are on the right machine.

Description: The compiler checks important software and hardware components to validate the license. If any of these components change, the license is no longer valid.

Solution: Restore the changed components or enter new license information.
License Errors

-6046  Cannot read license information. Please check FGLDIR and your environment.

Description: Several environment variables must be set correctly.

Solution: Check your environment variables (see the section “Required Environment Variables” on page 3-7). Check your license by running the program fglWrt -a see.

-6047  Bad information in license program. Please verify that you have installed the Dynamic 4GL license. For the Dynamic 4GL License Server, check that you are on the right machine.

Description: The compiler checks important software and hardware components to validate the license. If any of these components change, the license is no longer valid.

Solution: Restore the changed components or enter a new serial number.

-6048  Bad information in license program. Please verify that you have installed the Dynamic 4GL license. For the Dynamic 4GL License Server, check that you are on the right machine.

Description: The compiler checks important software and hardware components to validate the license. If any of these components change, the license is no longer valid.

Solution: Restore the changed components or enter a new serial number.

-6049  This is a runtime license. Compilation is not possible with it.

Description: You have a runtime license installed with this package. You cannot compile 4GL source code modules with this license.

Solution: If you want to compile 4GL source code, you need to purchase and install a development license. Contact your Dynamic 4GL distributor.

-6050  Temporary runtime license expired. Please contact your reseller.

Description: A license with a time limit has been installed and the license has expired.

Solution: Install a new license to activate the product. Contact your Dynamic 4GL distributor.
License Errors

-6051  Temporary runtime license expired. Please contact your reseller.
Description: A license with a time limit has been installed and the license has expired.
Solution: Install a new license to activate the product. Contact your Dynamic 4GL distributor.

-6052  Temporary runtime license expired. Please contact your reseller.
Description: A license with a time limit has been installed and the license has expired.
Solution: Install a new license to activate the product. Contact your Dynamic 4GL distributor.

-6053  FGLDIR environment value has changed. Please check it and give it the previous value.
Description: The value of FGLDIR or the location of FGLDIR has been changed.
Solution: Ask the person who installed the product for the location of the original installation directory and then set the FGLDIR environment variable.

-6054  Cannot open a license file for reading. Please check FGLDIR and your environment. Verify if you have installed the Dynamic 4GL license.
Description: The file containing the license is not readable by the current user.
Solution: Check that the FGLDIR environment variable is correctly set and that the file $FGLDIR/etc/f4gl.sn is readable by the current user.

-6055  Cannot open a license file for updating. Please check FGLDIR and your environment. Verify that you have installed the Dynamic 4GL license.
Description: The file containing the license cannot be overwritten by the current user.
Solution: Check if the FGLDIR environment variable is correctly set and if the current user can write to the file $FGLDIR/etc/f4gl.sn.
**License Errors**

- **-6056** Cannot write to a license file. Please check your permissions.
  
  Description: The file containing the license cannot be overwritten by the current user.
  
  Solution: Check that the `FGLDIR` environment variable is correctly set and that the current user can write to the file `$FGLDIR/etc/f4gl.sn`.

- **-6057** Cannot read a license file. Please verify if you have installed the Dynamic 4GL license.
  
  Description: The file containing the license cannot be read by the current user.
  
  Solution: Check that the current user can read the file `$FGLDIR/etc/f4gl.sn`. Also check that the `FGLDIR` environment variable is set correctly.

- **-6058** Bad format of a license file. Please verify if you have installed the Dynamic 4GL license.
  
  Description: The file containing the license has been corrupted.
  
  Solution: Reinstall the license. If you have a backup of the current installation of Dynamic 4GL, restore the files located in the `$FGLDIR/etc` directory.

- **-6059** Bad format of a license file. Please verify if you have installed the Dynamic 4GL license.
  
  Description: The file containing the license has been corrupted.
  
  Solution: Reinstall the license. If you have a backup of the current installation of Dynamic 4GL, restore the files located in the `$FGLDIR/etc` directory.

- **-6068** No Dynamic 4GL license installed.
  
  Description: There is no license installed for Dynamic 4GL.
  
  Solution: Install a license. If a license is already installed, check that the `$FGLDIR` variable is set correctly.
License Errors

-6069 Cannot uninstall the Dynamic 4GL license.
Description: There was a problem during the uninstall of the Dynamic 4GL license.
Solution: Check if the FGLDIR variable is correctly set in your environment and if the current user has permission to delete files in the $FGLDIR/etc directory.

-6070 You must specify fgllic.server if you want to use the Dynamic 4GL License Server.
Description: You are using the remote license process and you have set the value of fgllic.server, in $FGLDIR/etc/fglprofile, to localhost or to the 127.0.0.1 address.
Solution: You must use the real IP address of the machine even if it is the local machine.

-6071 Problem with directory %s. Check FGLDIR value and verify if this directory exists and has 777 mode.
Description: The compiler needs to make an operation in the specified directory.
Solution: Change the permission of this directory.

-6072 Cannot create file in directory %s. Check the FGLDIR value and verify that this directory exists and has 777 mode.
Description: The compiler needs to make an operation in the specified directory.
Solution: Change the permission of this directory to 777 mode.

-6073 Cannot change the mode of a file in directory %s. Verify that this directory exists and has 777 mode.
Description: The compiler needs to make an operation in the specified directory.
Solution: Change the permission of this directory to 777 mode.
License Errors

-6074 File %s does not have 777 mode or is not a directory. Check the mode with command `Is A $FGLDIR/lock`, or if you do not have any active users, use `rm -r $FGLDIR/lock`.

Description: The compiler needs to make an operation in the specified directory.

Solution: Change the permission of this directory. The $FGLDIR/lock directory contains only data needed at runtime by 4GL applications. When the application is finished, you can remove this directory. If you delete this directory while 4GL applications are running, the applications will be stopped immediately.

-6075 Cannot read from directory %s. Check the FGLDIR value and verify if this directory exists and has 777 mode.

Description: The compiler needs to make an operation in the specified directory.

Solution: Change the permission of this directory.

-6076 Bad lock tree. Please check your environment.

Description: There is a problem accessing the $FGLDIR/lock directory.

Solution: Check if the current user has sufficient permission to read and write to the $FGLDIR/lock directory. Check also if the FGLDIR environment variable is correctly set.

-6077 Bad lock tree. Please check your environment.

Description: There is a problem accessing the $FGLDIR/lock directory.

Solution: Check that the current user has sufficient permission to read and write to the $FGLDIR/lock directory. Check also that the FGLDIR environment variable is correctly set.

-6078 SYSEXERROR(%d)%s: Cannot set socket to non-blocking mode. Please check the system error message and retry your program.

Description: When starting an application, a problem occurs with the initialization of the socket of the Windows machine.

Solution: Restart the program. If the problem still exists, check that the TCP/IP stack is correctly installed and configured on your machine.
License Errors

-6079 Cannot read the name of the machine or cannot get the network IP address. With a Dynamic 4GL License Server, you must know the IP address of every Dynamic 4GL network client. Set FGLSERVER to the name or the IP address of your machine. It cannot be localhost.

Description: You are using the remote license process and you have set the value of fgllic.server, in $FGLDIR/etc/fglprofile, to localhost or to the 127.0.0.1 address.

Solution: You must use the real IP address of the machine even if it is the local machine. This is also true for the value used with the FGLSERVER variable.

-6080 Cannot get information from host %s. License server name may have a wrong value.

Description: The system cannot find the IP address of the specified host.

Solution: This is a configuration issue regarding your system. The command ping should not reply as well. Correct your system configuration and then try to execute your program.

-6081 Cannot ping host %s. Does not respond. License server name may have a wrong value. Check variable fgllic.server in your fglprofile file. Check your network configuration, or set fgllic.ping to a larger value.

Description: The license server cannot ping the client machine, or it does not get the response in the time limit specified by the fgllic.ping entry in the $FGLDIR/etc/fglprofile file.

Solution: Try to manually ping the specified machine. If this works, try to increase the value of the fgllic.ping entry in fglprofile. If the ping does not respond, fix the system configuration problem and then try the program again.

-6082 SYSERROR(%d)%s: Cannot set option TCP-NODELAY on socket. Please check the system error message and retry your program.

Description: There is a problem with the socket of the Windows machine.

Solution: Check that the system is correctly configured, and try the program again.
License Errors

-6083 SYSERROR(%d)%s: Cannot set option DONTLINGER on socket. Please check the system error message and retry your program.

Description: There is a problem with the socket of the Windows machine.

Solution: Check that the system is correctly configured, and try the program again.

-6084 SYSERROR(%d)%s: Cannot set option LINGER on socket. Please check the system error message and retry your program.

Description: There is a problem with the socket of the Windows machine.

Solution: Check that the system is correctly configured, and try the program again.

-6085 SYSERROR(%d)%s: Cannot connect to Dynamic 4GL License Server on machine %s. Please check following things:

- Value of variable fgllic.server
- If the license server machine is running
- If Dynamic 4GL service/daemon is running

Description: The application cannot check the license validity. To do so, it tries to communicate with the Dynamic 4GL license service running on the Windows NT machine where the product is installed.

Solution: Check that the Dynamic 4GL License Server is running on the machine where the product is installed.

-6086 SYSERROR(%d)%s: Cannot send data to Dynamic 4GL License Server. Please check the system error message and retry your program.

Description: There is a problem with the socket of the Windows machine.

Solution: Check that the system is correctly configured, and run the program again.

-6087 SYSERROR(%d)%s: Cannot receive data from Dynamic 4GL License Server. Please check the system error message and retry your program.

Description: There is a problem with the socket of the Windows machine.

Solution: Check that the system is correctly configured, and run the program again.
License Errors

-6088 You do not have permission to connect for the following reason: %s

Description: The program cannot connect to the license server because of the specified reason.

Solution: Try to fix the problem described and run your application again.

-6090 SYSERROR(%d)%s: Cannot create a socket to start the Dynamic 4GL License Server. Please check the system error message and retry your program.

Description: There is a problem with the socket of the Windows machine.

Solution: Check that the system is correctly configured, and run the program again.

-6091 SYSERROR(%d)%s: Cannot bind socket for the Dynamic 4GL License Server. Please check the system error message and retry your program.

Description: There is a problem with the socket of the Windows machine.

Solution: Check that the system is correctly configured, and run the program again.

-6092 SYSERROR(%d)%s: Cannot listen to the socket for the Dynamic 4GL License Server.

Description: There is a problem with the socket of the Windows machine.

Solution: Check that the system is correctly configured, and run the program again.

-6093 SYSERROR(%d)%s: Cannot create a socket to search an active client.

Description: There is a problem with the socket of the Windows machine.

Solution: Check that the system is correctly configured, and run the program again.

-6094 SYSERROR(%d)%s: WSAStartup gives previous error. Please check the system error message and retry your program.

Description: There is a problem with the socket of the Windows machine.

Solution: Check that the system is correctly configured, and run the program again.
License Errors

-6095 License not valid to run the Dynamic 4GL License Server: %s

Description: If you want to use the license server method with a Windows NT machine, your license must begin with the letters WLS.

Solution: Contact your Dynamic 4GL distributor to get a valid license.

-6096 Connection refused by Dynamic 4GL License Server.

Description: There is problem connecting the client machine to the Windows license server.

Solution: The problem is due to a configuration problem of the license server machine. Check the configuration of the machines and of the products.

-6098 Stopping the Dynamic 4GL License Server.

Description: The license server service is stopping.

Solution: This is an informational message.

-6099 SIGTERM received. Stopping the Dynamic 4GL License Server.

Description: The license server service is stopping.

Solution: This is an informational message.
Index

Numerics
4GL  3-3, 5-4, 5-5, 5-6, 5-7, A-2, A-7, B-2, B-6, C-2, C-7, C-8, E-3, E-5, Error-3, Error-5, Error-6, Error-7, Error-8, Error-11, Error-12, Error-13, Error-16, Error-20, Error-24

A
Arrays 5-4
Ataman C-13

B
BEFORE ROW 5-4
Boldface type Intro-5
Buttons C-10, E-8

C
compiler A-8, C-8, C-9, C-10
C language A-7
Canvas E-5
CC A-7, C-3
C-code Error-3, Error-4, Error-14
Channel Error-11
Chapter 2 - Microsoft Windows NT Installation 1-26
CHAR 5-6
cli-html.exe file 1-53
Comment icons Intro-6
confdesi E-1
Configuration C-13, E-1, E-2, E-6

D
Database C-5, Error-3, Error-4
DATE B-9
Datetime 5-4
DDE 5-6
Dependencies, software Intro-5
DISPLAY 5-6, 5-7, B-6, E-1, E-9, Error-6, Error-7, Error-8
Documentation, types of Intro-7

E
Environment variables Intro-5
ESQL/C C-3, C-4

F
fgl2c A-3, B-1, B-2, C-12, Error-7, Error-8, Error-13
fgl2p A-5, Error-3, Error-4, Error-14
FGLCC A-4
fglcl file 1-15
fglcl.conf file 1-19
fglcomp Error-3, Error-4, Error-5, Error-14
FGLDBPATH A-3, Error-3
FGLDEBUGON A-7

Configuration file for the HTML client 1-19
Contact information Intro-8
Important paragraphs, icon for Intro-6
INFORMIXDIR A-6, B-9, B-10, C-3, C-4, C-5, Error-15
INFORMIXSERVER B-9, B-10
INFORMIXSQLHOSTS B-9, B-10
Installation C-7, Error-4, Error-13
Installation of Web deployment components 1-53
Installation, of Web deployment option 1-14

KEY 5-5, B-25

Menu 5-5, B-16, B-30, E-4, E-5, E-6
Mouse 5-4

ns-httpd file 1-15
NT 3-3, 5-8, B-7, B-8, B-9, B-10, C-6, C-12

On-line manuals Intro-7

pager.42e 5-8
PATH A-4, A-9, A-10, Error-15, Error-16
P-code Error-3
Printed manuals Intro-7

Software dependencies Intro-5
System requirements
database Intro-5
software Intro-5