

Informix Dynamic 4GL

User Guide

Version 3.0
July 1999
Part No. 000-5412



Published by INFORMIX® Press

Informix Corporation
4100 Bohannon Drive
Menlo Park, CA 94025-1032

© 1999 Informix Corporation. All rights reserved. The following are trademarks of Informix Corporation or its affiliates:

Answers OnLine™; CBT Store™; C-ISAM®; Client SDK™; ContentBase™; Cyber Planet™; DataBlade®; Data Director™; Decision Frontier™; Dynamic Scalable Architecture™; Dynamic Server™; Dynamic Server™, Developer Edition™; Dynamic Server™ with Advanced Decision Support Option™; Dynamic Server™ with Extended Parallel Option™; Dynamic Server™ with MetaCube® ROLAP Option; Dynamic Server™ with Universal Data Option™; Dynamic Server™ with Web Integration Option™; Dynamic Server™, Workgroup Edition™; FastStart™; 4GL for ToolBus™; If you can imagine it, you can manage itSM; Illustra®; INFORMIX®; Informix Data Warehouse Solutions... Turning Data Into Business Advantage™; INFORMIX®-Enterprise Gateway with DRDA®; Informix Enterprise Merchant™; INFORMIX®-4GL; Informix-JWorks™; InformixLink®; Informix Session Proxy™; InfoShelf™; Interforum™; I-SPY™; Mediazation™; MetaCube®; NewEra™; ON-Bar™; OnLine Dynamic Server™; OnLine for NetWare®; OnLine/Secure Dynamic Server™; OpenCase®; ORCA™; Regency Support®; Solution Design LabsSM; Solution Design ProgramSM; SuperView®; Universal Database Components™; Universal Web Connect™; ViewPoint®; Visionary™; Web Integration Suite™. The Informix logo is registered with the United States Patent and Trademark Office. The DataBlade logo is registered with the United States Patent and Trademark Office.

Documentation Team: Mary Leigh Burke, Elaina Von Haas, Jennifer Leland, Mary Kraemer, Kathy Eckardt

Contributors: Jonathan Leffler

GOVERNMENT LICENSE RIGHTS

Software and documentation acquired by or for the US Government are provided with rights as follows:

- (1) if for civilian agency use, with rights as restricted by vendor's standard license, as prescribed in FAR 12.212;
- (2) if for Dept. of Defense use, with rights as restricted by vendor's standard license, unless superseded by a negotiated vendor license, as prescribed in DFARS 227.7202. Any whole or partial reproduction of software or documentation marked with this legend must reproduce this legend.

Table of Contents

Introduction

In This Introduction	3
About This Guide	3
Organization of This Guide	3
Types of Users	5
Software Dependencies	5
Documentation Conventions	5
Typographical Conventions	6
Icon Conventions	7
Additional Documentation	7
Informix Welcomes Your Comments	8

Chapter 1

Introducing Dynamic 4GL

In This Chapter	1-3
Introducing Dynamic 4GL	1-3
Windows Interface	1-3
Web Interface	1-4
Text Interface	1-4
GLS Support.	1-4
Dynamic 4GL Three-Tier Client/Server Architecture.	1-5
Differences Between Dynamic 4GL and 4GL	1-7
4GL Extensions	1-10
New Features in Dynamic 4GL	1-12
Main Features	1-12
Graphical Improvements	1-13
New 4GL Language Features in the 7.3 Release.	1-13

Chapter 2

Installing Dynamic 4GL

In This Chapter	2-3
Before Installing Dynamic 4GL	2-3
Upgrading Dynamic 4GL	2-4
Supported Operating Systems	2-4
Hardware Requirements	2-5
C-Compiler Requirements	2-6
Informix Client SDK	2-7
Dynamic 4GL Directory	2-7
Installing Dynamic 4GL on UNIX	2-8
Displaying the Installation Options	2-8
Installing Without a CD	2-9
Installing the Dynamic 4GL Files	2-10
GLS Installation	2-11
Licensing the Software	2-11
Compiling the Libraries	2-12
Creating the Environment Shell Script	2-13
Preparing to Install Dynamic 4GL on Windows NT	2-14
C-Compiler Requirement	2-14
Informix Database Server Requirement	2-14
TCP/IP Requirement	2-15
Hardware Prerequisite	2-15
Recommended Windows Client Prerequisite	2-15
Installing Dynamic 4GL on Windows NT	2-15
Dynamic 4GL Installation	2-15
Configuring Dynamic 4GL for Windows NT	2-18
Connecting to a Windows NT Database Server	2-19
Post-Installation Tasks	2-21
Installing and Configuring the Ataman Remote Login Service	2-22

Chapter 3

Basics of Using Dynamic 4GL

In This Chapter	3-3
Setting the Environment Variables	3-3
Compiling a Simple Program	3-4
Writing the Source Code	3-5
Compiling the Source Code	3-6
Compiling the Form-Specification File	3-7
Viewing the Dynamic 4GL Application	3-8

Chapter 4	Using the Dynamic 4GL Compiler	
	In This Chapter	4-3
	Setting Environment Variables for the Compiler	4-3
	Compiling Form-Specification Files and Help Message Files	4-4
	Compiling Form-Specification Files	4-4
	Compiling Help Message Files	4-5
	Generating a Database Schema File	4-5
	Compiling to P Code	4-6
	Overview of a P-Code Example	4-6
	Using C Functions in 4GL Applications	4-8
	Compiling to C Code	4-19
	Overview of a C-Code Example	4-19
	Using C Functions in 4GL Applications	4-21
	Compilation Tools	4-26
	Main Compilation Tools	4-27
	Other Compilation Tools	4-27
	Configuration Tools	4-28
	Miscellaneous Programs and Scripts	4-28
Chapter 5	Using Non-Graphical Extensions to 4GL	
	In This Chapter	5-3
	Channel Extensions	5-3
	Initializing Channel Extensions	5-4
	Opening a File	5-4
	Opening a Pipe	5-5
	Setting the Default Separator	5-6
	Reading Data from an Opened Channel	5-6
	Writing Data to a Pipe or Stream	5-7
	Channel Error Codes	5-8
	Sharing Information Using DDE	5-8
	Supported Windows Applications	5-9
	Using DDE Extensions	5-9
	Transmitting Values to a Windows Program	5-11
	Getting Values from a Windows Program	5-12
	Closing a DDE Connection	5-13
	Closing all DDE Connections	5-13
	Extending the DISPLAY ARRAY Statement	5-14

Returning Key Code Values	5-15
Returning Key Codes from P Code	5-16
Returning Key Codes from C Functions	5-18
Creating a Custom Character Filter	5-18
Starting a UNIX Emulator	5-19
Starting Windows Applications	5-20
Using Input Statement Functions	5-21
Returning a Value if a Field has been Modified	5-21
Returning the Name of a Field	5-23
Returning the Value of a Field	5-23
Setting the Value in a Field	5-23
Displaying a Row at a Given Line in a Screen Array.	5-24
Returning the Position of the Cursor	5-27
Terminating Applications	5-29
New Language Features	5-29
Enhanced SQL Syntax Support	5-30
Syntax for Expansion of Abbreviated Year Values	5-34
Enhanced Syntax for Screen Array Management	5-38
Dynamic Configuration of Report Output	5-46
New Built-In Operators	5-48
New Syntax to Hide the Comment Line	5-50
Editing Multibyte Data in 4GL Forms.	5-51
New Conditional Comments.	5-53

Chapter 6 Using Form Extensions to 4GL

In This Chapter	6-3
Implementing List Boxes	6-4
Implementing Buttons	6-6
Menu Buttons	6-6
Hot-Key Buttons	6-6
Buttons in the Form	6-9
Implementing Bitmaps	6-11
Implementing Check Boxes and Radio Buttons	6-11
Check Box Syntax	6-11
Radio Button Syntax	6-12
Invoking a Key Code	6-13
Combo Fields	6-14
Implementing Scrolling Fields	6-15
Creating Folder Tabs	6-16

Chapter 7

Using Graphical Extensions to 4GL

In This Chapter	7-3
Display Extensions	7-3
Calling Dynamic 4GL Libraries	7-3
Checking for UNIX or Windows	7-4
Checking for Windows Client Mode	7-5
Window-Management Functions	7-6
Setting the Default Size of a Window	7-6
Setting the Title of a Window	7-7
Retrieving Information from a Field.	7-8
Retrieving Information from an Application Window	7-8
Setting the Active Window	7-10
Closing a Window	7-10
Creating Toolbars and Toolbar Icons	7-11
Creating Dialog Boxes	7-12
Creating an Interactive Message Box	7-12
Displaying an Interactive Message Box.	7-14
Formatting Text in a Message Box	7-15
Entering a Field Value into a Message Box	7-16
Using Drawing Extensions	7-17
Mouse-Management Functions	7-18
Defining the Drawing Area.	7-19
Initializing the Drawing Function	7-20
Selecting a Drawing Area	7-20
Specifying the Text Insertion Point	7-21
Setting Line Width.	7-22
Clearing the Draw Function	7-22
Drawing Rectangles	7-23
Setting the Fill Color	7-23
Drawing an Oval	7-23
Drawing a Circle	7-24
Drawing a Line	7-24
Drawing Text	7-25
Drawing an Arc.	7-25
Drawing a Polygon	7-26

Chapter 8	Configuring the Dynamic 4GL Compiler	
	In This Chapter	8-3
	Configuring Dynamic 4GL	8-3
	Runtime Configuration File	8-4
	User Configuration File	8-4
	Program Configuration File	8-4
	General Configuration Settings	8-5
	Runtime Configuration Settings	8-6
	General Settings	8-6
	Graphical Daemon Autostart	8-9
	UNIX Settings	8-10
	Microsoft Windows Settings	8-11
	License Configuration Settings	8-14
	General Settings	8-14
	UNIX Settings	8-16
	GUI Settings	8-17
	General GUI Settings	8-17
	Menu GUI Settings	8-19
	Status Bar Settings	8-36
	Memory Mapping Settings	8-37
	Local Editing Settings	8-38
	Cut, Copy, and Paste Feature Settings	8-39
Chapter 9	Using the Configuration Manager	
	In This Chapter	9-3
	About the Configuration Manager	9-3
	Starting the Configuration Manager	9-3
	Starting on UNIX.	9-4
	Starting on Windows	9-4
	Using the Dynamic 4GL Configuration Manager	9-4
	File Menu	9-5
	Widget Menu	9-5
	The Help Menu	9-9
	How to Configure an Object with the Configuration Manager	9-10
	Opening a File.	9-10
	Configuration Types.	9-10
	The Different Configurations.	9-12

Chapter 10 Using the HTML Client

In This Chapter	10-5
Web Deployment Architecture	10-6
Why Deploy on the Web?	10-7
HTML Client Limitations	10-8
HTML Client Enhancements	10-9
Installing the HTML Client	10-9
Installing on UNIX	10-9
Installing on Windows NT	10-13
How Web Deployment Works at Runtime	10-16
Supplying Your Own Headers and Footers	10-19
Disabling Password Display	10-19
Similarities Between a .per File and an .html File	10-19
Deploying a Sample Application	10-20
Screens.	10-22
Step 1: Creating a Dynamic 4GL Application.	10-22
Step 2: Editing the Server Configuration File.	10-23
Step 3: Creating a Script to Initialize the Application	10-31
Step 4: Editing Your Client Configuration File	10-31
Step 5: Starting the HTML Server Process on UNIX	10-31
Step 6: Starting the Browser	10-32
Step 7: Using the Application	10-32
Step 8: Enhancing the Application	10-36
Creating Email and Web Site Links	10-36
Enhancing the Screen Files	10-37
Horizontal split	10-39
Table	10-39
How Links Between Pages Work	10-40
HTML Emulation for Tables	10-41
Dynamic 4GL Features	10-41
Security Levels	10-44
Default Security	10-44
Recommendations for Enhancing Security	10-46
Application, Web Server, and Database Security	10-46
Preventing Security Problems	10-47
Configuring the Web Deployment Software	10-48

Configuration Settings in the fglcl.conf file	10-48
Location	10-48
fglserver	10-49
debug	10-49
HTMLdebug	10-50
Security	10-50
Security Through the Web Server	10-51
Security Through the File System	10-51
Summary	10-52
Configuring the appname.conf File	10-52
General Configuration Settings	10-52
Pre and Post Messages	10-56
Styles	10-59
Spawning	10-63
Arrays	10-65
Troubleshooting the UNIX Installation	10-66
Checking the HTML Client	10-66
Checking the HTML Server	10-68
Manual Installation on UNIX	10-69
Extracting the Files	10-69
Installing the HTML Client on the Web Server	10-70
Installing the HTML Server on the Application Server	10-71
Installing the HTML Documentation on the Web Server	10-72
Installing the Example	10-72
Troubleshooting the Windows NT Installation	10-73
Checking the HTML Client	10-73
Checking the HTML Server	10-73

Chapter 11 Using the Java Client

In This Chapter	11-3
Introduction	11-3
Programs and Applets	11-4
Swing	11-5
Server-Side Components	11-5
How Dynamic 4GL Uses Java	11-5
Java Client Limitations	11-8
Java Client Security	11-8
Java Client Definitions	11-8

Requirements	11-11
Java Client Web Browser Requirements	11-11
Client Java Applet Viewer Requirements	11-12
Web Server Hardware and Software Requirements	11-13
Dynamic 4GL Application Server Requirements	11-13
Installing the Java Client	11-14
UNIX Installation	11-14
Windows NT Installation	11-20
Additional Installation Tasks	11-23
Configuring the Java Client	11-37
Editing the cjac.cnf File	11-38
Sample cjac.cnf file.	11-45
Editing the clijava.cnf File	11-49
Running an Application with the Java Client	11-54
Creating the HTML Page	11-54
Setting CJA Parameters	11-55
Running the Application	11-56
Java Client Enhancements	11-57

Chapter 12 Using the Windows Client

In This Chapter	12-3
Windows Client Architecture	12-3
Windows Client Requirements	12-4
Dynamic 4GL Server Requirements	12-5
Installing the Windows Client.	12-5
Installing the Windows Client on a Network.	12-8
Starting and Configuring the Windows Client	12-9
Starting the 4GL Server	12-9
Creating a Connection	12-9
Connection Checking.	12-11
Windows Client Language	12-13
Setting the Server Environment Variables	12-14
Using the VGA Driver with Windows 3.1	12-15
Running the Windows Client Example	12-15
Configuring the Environment Variables	12-17
Starting a P-Code Application.	12-18
Authorizing the Client Computer	12-18
Starting a C-Code Application.	12-19
Successful Connection	12-19

Security Features	12-20
Authorizing a Connection.	12-20
Connecting Without a Password	12-21
Command-Line Features	12-22
Special Tags Features	12-22
ilogin Command-Line Features	12-24
Customizing the Login Dialog Box	12-27
Using Ataman Remote Connection Services	12-29
Adding a Scrollbar to the Terminal Emulation Window	12-30
System Colors	12-31
Customizing the Windows Client Installation	12-31
Customizing Icons, Titles, and Directories	12-32
Specifying the Windows Client Icons	12-32
Installing Documentation	12-36
Configuration Files	12-37
Configuration File (WTKSRV.INI) Entries	12-37
Splash Screen Configuration	12-43
User-Defined Configuration File	12-47
User-Definable WTKSRV.INI Entries	12-48
Winframe from CITRIX	12-50
First Method	12-50
Second Method	12-52

Chapter 13 Using the X11 Client

In This Chapter	13-3
UNIX X11 Client Configuration	13-3
Installing the X11 Client	13-4
Managing Application Windowing	13-5
Running the Program on the X11 Client	13-8

Appendix A Environment Variables

Appendix B Common Problems and Workarounds

Appendix C Error Messages

Appendix D Global Language Support

Index

Introduction

In This Introduction	3
About This Guide	3
Organization of This Guide	3
Types of Users	5
Software Dependencies	5
Documentation Conventions	5
Typographical Conventions	6
Icon Conventions	7
Additional Documentation	7
Informix Welcomes Your Comments	8

In This Introduction

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

About This Guide

The *Informix Dynamic 4GL User Guide* describes how to develop 4GL applications on UNIX and Windows NT, and then deploy them, in either graphical or text mode, on various platforms, such as Windows 95, Windows NT, and X11 Window System clients.

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

Organization of This Guide

This guide includes the following chapters:

- [Chapter 1, “Introducing Dynamic 4GL,”](#) introduces Dynamic 4GL.
- [Chapter 2, “Installing Dynamic 4GL,”](#) describes how to install Dynamic 4GL on both UNIX and Windows.
- [Chapter 3, “Basics of Using Dynamic 4GL,”](#) explains how to set environment variables and compile a simple program in Dynamic 4GL.
- [Chapter 4, “Using the Dynamic 4GL Compiler,”](#) describes how to compile various types of files and how to compile to P code or C code.

- [Chapter 5, “Using Non-Graphical Extensions to 4GL,”](#) describes the features that have been added that extend the functionality of 4GL. These features do not affect the graphical interface.
- [Chapter 6, “Using Form Extensions to 4GL,”](#) describes the features that have been added that extend the functionality of 4GL forms.
- [Chapter 7, “Using Graphical Extensions to 4GL,”](#) describes the features that have been added that extend the functionality of 4GL. These features affect the graphical user interface (GUI).
- [Chapter 8, “Configuring the Dynamic 4GL Compiler,”](#) describes how to edit the **fglprofile** file to change the behavior of the Dynamic 4GL compiler.
- [Chapter 9, “Using the Configuration Manager,”](#) describes how to use the Configuration Manager to change the look of a GUI.
- [Chapter 10, “Using the HTML Client,”](#) describes how to deploy your Dynamic 4GL applications on a Web server and enhance the appearance of the Dynamic 4GL application for display with a Web browser.
- [Chapter 11, “Using the Java Client,”](#) describes how to install, configure, and execute the Java Client.
- [Chapter 12, “Using the Windows Client,”](#) describes how to install, configure, and execute the Windows Client.
- [Chapter 13, “Using the X11 Client,”](#) describes how to install, configure, and execute the X11 Client.
- [Appendix A, “Environment Variables,”](#) describes the environment variables you need to know about to use Dynamic 4GL.
- [Appendix B, “Common Problems and Workarounds,”](#) provides workarounds for common problems that you might encounter.
- [Appendix C, “Error Messages,”](#) lists the messages in numerical order and describes how to correct the situation that initiated the error message.
- [Appendix D, “Global Language Support,”](#) describes how to use the GLS features with Dynamic 4GL.

Types of Users

This guide is written for all Dynamic 4GL users.

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

Documentation Conventions

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

The following conventions are covered:

- Typographical conventions
- Icon conventions

Typographical Conventions

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




Convention	Meaning
KEYWORD	All primary elements in a programming language statement (keywords) appear in uppercase letters in a serif font.
<i>italics</i> italics <i>italics</i>	Within text, new terms and emphasized words appear in italics. Within syntax and code examples, variable values that you are to specify appear in italics.
boldface boldface	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.
<code>monospace</code> <i>monospace</i>	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
→	This symbol indicates a menu item. For example, “Choose Tools→Options ” means choose the Options item from the Tools menu.



Tip: 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.

Icon	Label	Description
	<i>Warning:</i>	Identifies paragraphs that contain vital instructions, cautions, or critical information
	<i>Important:</i>	Identifies paragraphs that contain significant information about the feature or operation that is being described
	<i>Tip:</i>	Identifies paragraphs that offer additional details or shortcuts for the functionality that is being described

Additional Documentation

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

- **Printed documentation.** The *Informix Dynamic 4GL User Guide* is available as a printed guide.
- **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.** Some additional documentation about Web server configuration is provided in supplementary HTML files. See [“Installing the HTML Documentation on the Web Server”](#) on [page 10-72](#) 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 guide that you are using
- Any comments that you have about the guide
- Your name, address, and phone number

Write to us at the following address:

Informix Software, Inc.
SCT Technical Publications Department
4100 Bohannon Drive
Menlo Park, CA 94025

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

`doc@informix.com`

We appreciate your suggestions.

Introducing Dynamic 4GL

In This Chapter	1-3
Introducing Dynamic 4GL	1-3
Windows Interface	1-3
Web Interface	1-4
Text Interface	1-4
GLS Support.	1-4
Dynamic 4GL Three-Tier Client/Server Architecture.	1-5
Dynamic 4GL Client Example	1-6
Dynamic 4GL Architecture	1-6
Differences Between Dynamic 4GL and 4GL	1-7
4GL Extensions	1-10
New Features in Dynamic 4GL.	1-12
Main Features	1-12
Graphical Improvements	1-13
New 4GL Language Features in the 7.3 Release.	1-13

In This Chapter

This chapter introduces you to the features of Dynamic 4GL, including the differences between the 4GL compiler and the Dynamic 4GL compiler.

Introducing Dynamic 4GL

Dynamic 4GL allows you to recompile your 4GL source code, transforming your existing text-based applications into a thin client/server system that can display your 4GL application with a graphical user interface (GUI).

In addition to your 4GL routines, you can enhance your applications for display with a GUI using Dynamic 4GL extensions. For example, you might add check boxes or list boxes to your GUI by enhancing your current 4GL source code.

Windows Interface

The GUI is displayed by a graphics server running on the client. The graphics server can be either X11 or a Windows version of the Tcl/Tk software, called WTK, which is provided with the Dynamic 4GL software.

By changing a single environment variable, you can also execute your programs in ASCII mode. This means that users access your programs in the same way they would access 4GL programs—by logging on to the same computer that runs the programs. The ease by which you can change modes allows you to control the migration rate of your client computers.

Web Interface

You can also provide a GUI for your Dynamic 4GL applications through any compatible Web browser. You can use either HTML or Java to do this. To use HTML, you run a daemon provided with the Dynamic 4GL software that converts the output of your program to HTML. To use Java, you must have a Web server that supports servlets. The 4GL source code does not need to be converted to use either method.

Text Interface

You can, of course, continue to display your Dynamic 4GL applications using text or ASCII.

GLS Support

Dynamic 4GL supports Informix Global Language Support (GLS). The GLS feature allows Informix database servers to handle different languages, cultural conventions, and code sets.

While Dynamic 4GL is fully compliant with Informix GLS, there are some restrictions. You cannot use GLS with the Dynamic 4GL HTML or X11 Client. In addition, you must be using 4GL 7.2 or later (this version is the first 4GL version to support GLS).

Dynamic 4GL Three-Tier Client/Server Architecture

By a simple recompilation with Dynamic 4GL, the 4GL sources are transformed into a three-tier client/server system, as [Figure 1-1](#) shows. This system facilitates migration from ASCII-based terminals to systems with a GUI.

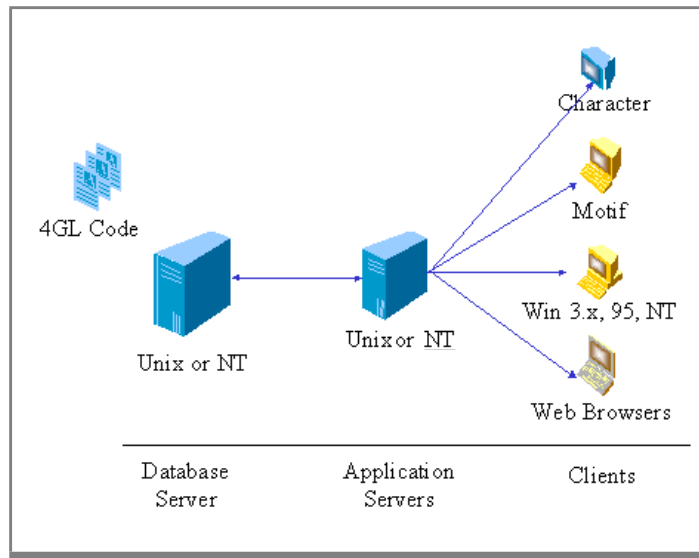


Figure 1-1
4GL Three-Tier
Client/Server
Architecture

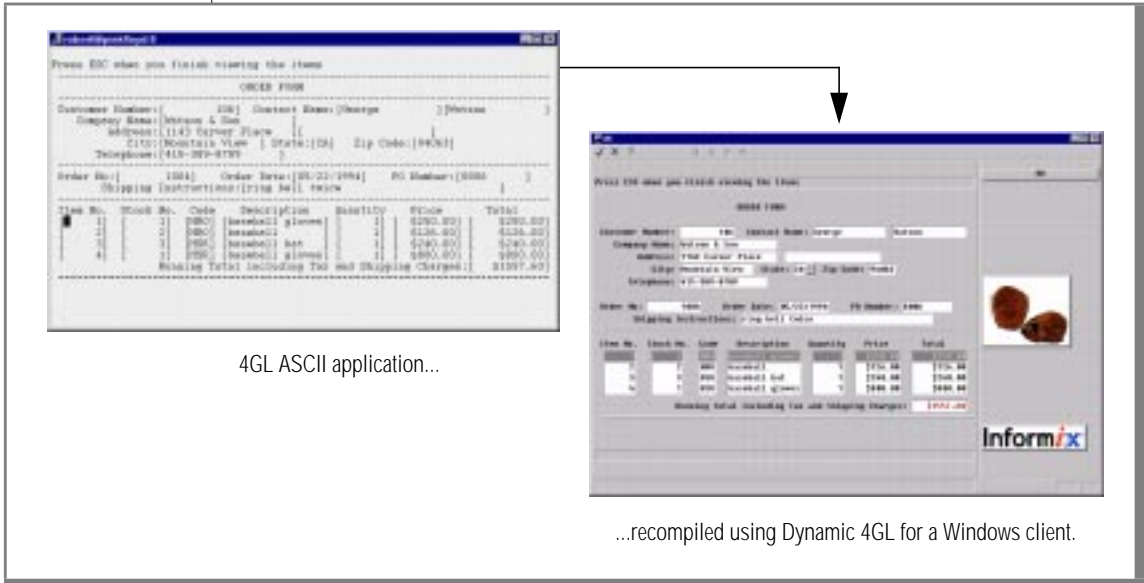
Dynamic 4GL uses this architecture in the following ways:

- The 4GL application is compiled in P code and deployed on the application server where a Dynamic Virtual Machine (DVM) is installed.
- Dynamic 4GL can access a database server anywhere on the network.
- Dynamic 4GL can be accessed by ASCII, X11, or Windows clients.
- A Web server can access Dynamic 4GL.
- The Java and HTML clients can access Dynamic 4GL through the Web Server.

Dynamic 4GL Client Example

Figure 1-2 shows a 4GL application before and after it was converted to a Windows client.

Figure 1-2
4GL Application Converted
to a Windows Client



Dynamic 4GL Architecture

The application server can be a UNIX server or a Windows NT computer. In a typical installation, Dynamic 4GL is installed on the application server (either the development or runtime version) with the 4GL programs. The database server is usually installed on this same computer, but this is not required.

The client computers can be X11 compliant interfaces or Windows computers. Each client has its own Dynamic 4GL daemon that handles the GUI aspects of 4GL applications.

Differences Between Dynamic 4GL and 4GL

Dynamic 4GL is based on the features in Version 7.3 of 4GL. However, it is possible to compile programs created with older versions of 4GL.

The following list summarizes the differences between Dynamic 4GL and 4GL:

- **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 Rapid Development System (RDS).

- **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 Dynamic 4GL 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 **fglrun.arrayIgnoreRangeError** in the configuration files if you want the compiler to continue execution.
- **Input array.** If you use an **INPUT** or **INPUT ARRAY** instruction in Dynamic 4GL, be sure that at least one field is not declared as **noentry**.

The following example shows 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.** Creating graphical applications in Dynamic 4GL makes 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**.

Right-clicks and double-clicks are also supported. The following two entries in the configuration files accommodate these mouse actions:

```
gui.key.doubleClick.left= "key" # default is KEY_accept
gui.key.click.right      = "key" # default is F36
```

- **Reports.** Normally, if any value that is part of an aggregate function (**avg,sum,...**) is NULL, the result of the function is also NULL. In Dynamic 4GL reports, you can have aggregates return 0 in such a case by setting 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 **fglrun.cursor.global** entry in the configuration file. This choice is made at runtime with Dynamic 4GL.
- **Menu.** In Dynamic 4GL, 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:

```
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:

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

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

```
DATABASE dbname
```

- **PROMPT.** In Dynamic 4GL, the 4GL PROMPT statement waits for an answer in the graphical window at the prompt line if the graphical window is opened first. If no graphical window is open before the PROMPT statement is executed, the prompt is made in the ASCII terminal, for example:

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

- In the previous example, PROMPT is executed in the terminal and in terminal mode even if you are in graphical mode.
- The following example shows how to execute PROMPT in graphical mode:

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

4GL Extensions

The following list summarizes the features that were added to the 4GL language:

- **Channel functions.** Channel functions are 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.** DDE functions 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 computer or on a UNIX computer but from a Windows front end.
- **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 or disappear when the program using it is finished.
- **Canvas functions.** This set of functions allows you to draw basic shapes in 4GL 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, 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.
- **New triggers for the DISPLAY ARRAY statement.** The DISPLAY ARRAY statement now accepts [BEFORE | AFTER | EXIT | CONTINUE] DISPLAY and [BEFORE | AFTER] ROW statements.
- **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. To switch to the previous or next page, click a button.

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

The **pager.42e** command starts the graphical editor used to display reports in C code. The **fglpager** command starts the graphical editor used to display reports in P code. The same pager can be used from the UNIX prompt for the C version:

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

or on Windows NT for the P-code version:

```
$ fglpager [filename]
```

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

- ❑ 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.

New Features in Dynamic 4GL

The following features are new to Dynamic 4GL.

Main Features

Dynamic 4GL provides GLS and Java client support, as follows:

- **GLS support.** Allows Informix database servers to handle different languages, cultural conventions, and code sets. For more information, refer to [Appendix D, “Global Language Support.”](#)
- **Java client.** Allows 4GL applications to be displayed as a Java applet within a browser with little or no recoding. For more information, refer to [Chapter 11, “Using the Java Client.”](#)

Graphical Improvements

Dynamic 4GL has the following graphical feature enhancements:

- **Local Editing.** Enables the client to wait until an entire value has been entered into a field before submitting the information to the application server. For more information, refer to [“Local Editing Settings” on page 8-38](#).
- **Cut and Paste.** Enables standard cut and paste functionality within the Windows Client or X11 Client. For more information, refer to [“Cut, Copy, and Paste Feature Settings” on page 8-39](#).
- **Folder Tabs.** Allows multiple screen forms to be displayed using folder tabs (appear similar to Windows folder tabs) enabling the user to easily navigate between multiple screens. For more information, refer to [“Creating Folder Tabs” on page 6-16](#).
- **New terminate handler.** Allows a standard close window option to appear in the upper-right corner of a GUI window. Alt+F4 will also close the window.
- **Control Frame management.** Allows a control frame to be anchored to the right or left of the screen.
- **Status bar.** Allows the status of certain keys to be displayed on the status bar. For more information, refer to [“Status Bar Settings” on page 8-36](#).
- **System color on windows.** Allows the Microsoft Windows system settings to determine the colors displayed by an application. For more information, refer to [“System Colors” on page 12-31](#).
- **Splash screen.** Allows a splash screen to be displayed after starting an application. For more information, refer to [“Splash Screen Configuration” on page 12-43](#).

New 4GL Language Features in the 7.3 Release

The following list shows the new 4GL 7.3 language features.

- **DBCENTURY fields.** Provides year 2000 support.
- **SQL Grammar extension.** Supports 4GL SQL syntax for versions 4.1 to 7.3 (previously only 4.1 was supported).

- **Syntax cleaning.** Provides easier names for many Dynamic 4GL functions (although old function names will still be supported).
- **New terminate signal.** Allows a 4GL application to send a terminate signal (available only for UNIX).
- **New synonym for the concatenation string operator.** Provides support for the || concatenation string operator.
- **Control INSERT and DELETE operations in INPUT ARRAY.** Allows the INSERT and DELETE keys to be enabled or disabled independent of each other.
- **New ATTRIBUTE format in INPUT ARRAY.** Supports for two new functions to set attribute formats inside a dialog box.
- **Program-controlled INSERT and DELETE operations.** Allows one or more rows of data to be inserted into the middle of a program array.
- **Program override of INSERT and DELETE operations.** Allows INSERT and DELETE keys to be overridden even if they are enabled.
- **Dynamically configure size of report.** Allows the size of the report page and report destination to be specified when a report starts.
- **Dynamic control of the effective size of a program array (INPUT ARRAY).** Allows the size of the INPUT ARRAY to be dynamically controlled.
- **Current row highlighted automatically.** Allows the current row to be highlighted without requiring any code changes.
- **Get size of screen array.** Declares the size of a named screen array so that the correct number of values is displayed.
- **COMMENT OFF in windows.** Allows a window with a form to be reduced to one or two (an input and comment line) lines.

For more information, see [“New Language Features” on page 5-29](#) and the *INFORMIX-4GL Reference Manual*.

Installing Dynamic 4GL

In This Chapter	2-3
Before Installing Dynamic 4GL.	2-3
Upgrading Dynamic 4GL	2-4
Supported Operating Systems.	2-4
Hardware Requirements	2-5
TCP/IP Requirements	2-5
Disk Space Requirements	2-6
C-Compiler Requirements	2-6
Informix Client SDK	2-7
Dynamic 4GL Directory	2-7
Installing Dynamic 4GL on UNIX.	2-8
Displaying the Installation Options	2-8
Installing Without a CD	2-9
Installing the Dynamic 4GL Files.	2-10
GLS Installation	2-11
Licensing the Software	2-11
Licensing After 30 Days	2-12
Avoiding Licensing on Reinstall.	2-12
Compiling the Libraries	2-12
Creating the Environment Shell Script	2-13
Preparing to Install Dynamic 4GL on Windows NT.	2-14
C-Compiler Requirement	2-14
Informix Database Server Requirement	2-14
TCP/IP Requirement.	2-15
Hardware Prerequisite	2-15
Recommended Windows Client Prerequisite	2-15

Installing Dynamic 4GL on Windows NT	2-15
Dynamic 4GL Installation	2-15
Configuring Dynamic 4GL for Windows NT	2-18
Connecting to a Windows NT Database Server	2-19
Post-Installation Tasks	2-21
Installing and Configuring the Ataman Remote Login Service	2-22

In This Chapter

This chapter describes how to install the Dynamic 4GL development package. This chapter includes instructions for installing the application server on either UNIX or Windows NT. For directions on how to install a client (Windows or X11) and display a Dynamic 4GL application with a GUI, (Java, HTML, Windows, or X11), refer to the chapters later in this manual.

Before Installing Dynamic 4GL

Before you install the Dynamic 4GL development package, review the following list. For more information, you can refer to the sections that follow the list.

- **Dynamic 4GL upgrades.** If you are upgrading from a previous version of Dynamic 4GL, unset any Dynamic 4GL environment variables before you install the upgrade.
- **Supported operating system.** Check to be sure that Dynamic 4GL supports your operating system.
- **Hardware requirements.** Check that your system meets the minimum hardware requirements. Be sure that your computer has enough disk space to install Dynamic 4GL and a TCP/IP connection.
- **C-compiler requirements.** You must have a compatible C compiler installed even if you do not plan on compiling to C code. The C compiler creates the P-code runner. The `PATH` environment variable should include the location of the C compiler.
- **Informix Client Software Developer's Kit.** You should have Informix Client SDK 2.01 (or later) installed if you want to use the GLS features. You install the Client SDK separately from Dynamic 4GL.

- **Dynamic 4GL directory.** You should install Dynamic 4GL in its own directory to make it easier to upgrade or maintain the software.
- **Informix database server.** It is recommended that you have the database server running to check the success of the installation. You will also need access to the database server to create the P-code runner.

Upgrading Dynamic 4GL

If you are upgrading your version of Dynamic 4GL, unset the **FGLDIR** environment variable (or set it to the new directory.) If you are changing the installation of the Informix database or the Client SDK, unset the following environment variables as well:

- **FGLDBS**
- **FGLLIBSQL** and **FGLLIBSYS** (if you are compiling to C code)

Supported Operating Systems

You can install Dynamic 4GL on the following operating systems, however, some operating systems do not support GLS, as the following table illustrates.

Important: For information about any changes to the supported operating systems, refer to the Dynamic 4GL **ReadMe** file.



Operating System	Version	GLS Support
AIX	3.2.5	No
	4.2.1 & higher	Yes
DG/UX	R4.20 & higher (Intel)	Yes
HPUX	10.01 & higher	Yes
SCO UNIX	3.2.5.0.x (ELF)	Yes
Unixware	2.10.x	Yes
	7	Yes

(1 of 2)

Operating System	Version	GLS Support
Digital UNIX (OSF)	4.0	Yes
IRIX	5.3 (o32 format)	Yes
	6.2 & higher (n32 format)	Yes
Reliant UNIX (SINIX)	5.43 & higher	Yes
SUN	SPARC Solaris 2.5.1 & higher	Yes
	Intel Solaris 2.6 & higher	Yes
Windows	NT4 (Intel)	Yes
Dynix/Ptx	4.4.2 & higher	Yes
Linux	kernel 2.0.3 & higher	Yes
	glibc 2.0.7 & higher	Yes

(2 of 2)

Hardware Requirements

Before you install Dynamic 4GL, check that your system meets the following requirements. A network card is required.

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



TCP/IP Requirements

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. For example, on SCO UNIX System 3.2, the library **libsocket.a** must be present. For Windows NT, use the Microsoft TCP/IP stack.

Disk Space Requirements

The following disk space is required for installation:

- 15 megabytes of disk space in the `/tmp` directory (or another specified directory). This disk space is released after the installation is complete.
- 20 megabytes of disk space for the compiler and the `Tcl/Tk` graphical display server.

C-Compiler Requirements

You must have an ANSI-compatible C compiler on the development computer (even if you do not plan on compiling to C code). The C compiler is used during the installation to create a runner that links the following libraries:

- System libraries
- INFORMIX-ESQL/C libraries (available in the Client SDK)
- Dynamic 4GL libraries

If you want to use a native C compiler, check that it conforms to ANSI standards. An ANSI-compatible C compiler must accept the `-c` flag to produce object files and the `-o` flag to produce executable files.

Also, check to be sure that the path to the C compiler is added to the `PATH` environment variable setting. If you do not want to use your native C compiler, install the GNU C compiler delivered with Dynamic 4GL.



Important: *You need to create a new runner whenever one of the following components changes: your operating system, database interface, or version of Dynamic 4GL.*

Be sure that you can compile ESQL/C programs with the ESQL/C compiler. If you are not using the default C compiler (which is normally `cc`), make sure that you set the `INFORMIXC` environment variable to the compiler you are using, such as `gcc` (the GNU C compiler), as well as the documented `FGLCC` and `CC` environment variables, as follows:

```
INFORMIXC=gcc
export INFORMIXC
```


If you are using the GNU C compiler (GCC) from the product CD, ensure that you have both installed GCC before you install the Dynamic 4GL compiler and set the GCC environment variable correctly with the **envgcc** script.

For more information about the GCC environment variable, see “GCC Environment Variables” on page A-7.

Informix Client SDK

To use GLS features, you should have Client SDK 2.01 (or later) installed prior to installing Dynamic 4GL. If you do not need the GLS libraries, the Client SDK is recommended but not required.

If you are using an older version of Informix ESQL/C and you are not planning on using GLS, you should still be able to install Dynamic 4GL and create the necessary runners. Some features (for example, EXECUTE IMMEDIATE) are not available in 4.1x ESQL/C but are available in later versions. If possible, consider using ESQL/C 5.1x, 7.2x or the Client SDK versions of ESQL/C.

You can download Informix Client SDK from the following Informix Web site. In addition, the Web site provides installation instructions.

<http://www.intraware.com/informix/>

Dynamic 4GL Directory

You should install Dynamic 4GL in a separate directory. You will find this makes upgrading or maintaining Dynamic 4GL easier.



Important: You can install the client display server in the same directory as the Dynamic 4GL compiler. However, if you expect to install a compiler license and a runtime license for Dynamic 4GL on the same computer, you should keep the display server code separate.

Installing Dynamic 4GL on UNIX

The following steps start the installation. The Dynamic 4GL installation program uses Bourne shell scripts. During the installation, you will be prompted to perform the following steps:

- If necessary, back up a previous version of Dynamic 4GL.
- Install all the needed files on your system.
- Create a default environment for compilation.
- Install the GLS components.
- License the compiler (or keep the previous license for an update).
- Create the P-code runner and needed libraries and tools.



Tip: If you have problems installing Dynamic 4GL, you can perform a manual installation. For instructions, see [“Installing the Dynamic 4GL Software Manually”](#) on page B-1.

Displaying the Installation Options

Display the Dynamic 4GL online installation directions using the `-h` flag. For example, enter the following command:

```
$ /bin/sh ./sco-dev-2.13.019.sh -h
```

Installing from the Dynamic 4GL CD

If you have a CD accessible from UNIX, perform the following steps:

1. Log on as user root.
2. Mount the Dynamic 4GL CD.

3. Change to the mount directory.
4. Run the shell script named **install.sh** with the following command:

```
$ /bin/sh ./install.sh -i product type
```

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

Product Type	Description
compiler	Installs the development package, including all the tools needed to compile and execute your 4GL programs.
runtime	Installs a runtime package. The runtime package allows you to execute previously compiled Dynamic 4GL programs, but it does not allow you to compile them.
patch	Installs a patch over a version
demo	Installs the trial package

If you do not have a **/tmp** directory or do not have enough space on your **/tmp** directory for Dynamic 4GL, you will need to direct the installation script to another directory using the **-w** flag. For example:

```
$ /bin/sh ./sco-dev-2.13.019.sh -i compiler -w /usr/tmp
```

Installing Without a CD

If you do not have a CD accessible from UNIX, copy the file located in the directory **/OS/UNIX/your_OS_name/SELFEXTR/** that corresponds to the package you want to install. For example, the scripts for a Solaris workstation are:

```
OS/UNIX/SUN/SELFEXTR/COMPILER.SH
OS/UNIX/SUN/SELFEXTR/RUNTIME.SH
OS/UNIX/SUN/SELFEXTR/GNUC.SH
OS/UNIX/SUN/SELFEXTR/TCLTK.SH
```

Then log on, go into the directory where you copied the package, and run the following command:

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

Use the scripts as follows to install the various packages:

- To install the development system, run **COMPILER.SH**.
- To install the runtime system, run **RUNTIME.SH**.
- To allow use of an X11 client with either the development system or the runtime system, run **TCLTK.SH** in addition to the system installation script.
- To install the GNU C compiler, run **GNUC.SH** before installing either the development system or the runtime system, and set your environment to point to the location of the compiler.

You need the GNU C compiler only if you do not already have an ANSI C compiler on your system.

This shell creates all the necessary files and starts the installation process. For more information, refer to “[Post-Installation Tasks](#)” on page 2-21.

Installing the Dynamic 4GL Files

The installation determines the host-operating system and checks that 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 do not login as superuser, you get a warning that some administrative operations will be skipped. However, the administrative operations do not affect the operation of Dynamic 4GL.

You will also be prompted to use the default values. If you might want to change any of the default values, enter **NO**. For instance, if you want to change the directory where Dynamic 4GL is installed, you would enter **NO**.

If you want to overwrite an installed version of the compiler with a new version, you are prompted for the creation of a backup archive of the existing compiler.

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

GLS Installation

During the installation, you will be prompted to install Dynamic 4GL in GLS enhancements. You should only install the GLS enhancement if you are planning on using the GLS features. The GLS enhancement will install additional files. For more information on GLS, refer to [Appendix D, “Global Language Support.”](#)



Important: Remember, you must have the Client SDK 2.01 (or later) installed and the database server running to install the GLS features.

Licensing the Software

After Dynamic 4GL has installed all the application files, you will be prompted to license the software. To license Dynamic 4GL, you need the serial number and serial number key supplied with in the Dynamic 4GL package. You also need access to the Internet and a web browser.

You will be prompted for the serial number provided with the Informix media. The serial number follows the Informix format—an 11-character alphanumeric string. Type it and press RETURN, as the following example shows:

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

Then enter the 12-character serial number key provided with the Informix media and press RETURN. After this operation, the installation number is generated.

```
Enter your serial number KEY (uppercase letters and numerals) >
QR7CNNANJ8VI
Your installation NUMBER is "EZ0A8MSHEADC (1)".
```

You then need to display the Informix On-Line Licensing website located at the following address:

```
http://www.informix.com/keyissue
```

Follow the website directions to generate an Installation key to complete the licensing.

Enter the installation key to complete the licensing and press RETURN.

```
Do you want to give the installation KEY now (y/n)? y
Enter the installation KEY (call your vendor to obtain it) > 9PF6DKCAUTCU
License installation successful.
```

Licensing After 30 Days

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.*

Avoiding Licensing on Reinstall

If you reinstall the software, you can avoid entering a new serial number key. To do this, reinstall the product into the same **FGLDIR** directory (either physically or using a logical link).

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

```
$ fglWrt -d
```

Compiling the Libraries

If you want to compile to C code or create a custom P-code runner, you need to install the appropriate libraries.

You will be prompted to create the C-code libraries:

```
Do you want to create the C code libraries:
Options:( [Y]es | [N]o | [C]ancel)
Default:[N]y
```



When prompted, answer **Yes** to begin to create the P-code libraries:

```
Do you want to create the p-code libraries:
Options: ( [Y]es | [N]o | [C]ancel)
Default: [Y]
```

Dynamic 4GL automatically attempts to link a P-code runner and creates two script files in the installation directory:

- **envcomp** (Bourne shell)
- **envcomp.csh** (C shell)

If the P-code runner was not successfully created, it means your system is not configured correctly. Check that you have the required software installed and configured. To try to create the P-code runner again, run the following script file and correct any problems the script reveals:

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

You can continue to run **findlib.sh** until you have created the P-code runner.

***Important:** If you do not have INFORMIX-ESQL/C or INFORMIX-4GL (compiled) currently installed, a version without the database interface (**fglnodb** runner) is installed. In this case, you can use this runner to execute a compiled 4GL program provided it does not contain SQL statements. If you try to execute an SQL statement, it will generate an error.*



Creating the Environment Shell Script

After licensing the software, Dynamic 4GL prompts you to create an environment file called **envcomp**. This shell script sets up the main environment variables required for using Dynamic 4GL.

```
Do you want to create an environment file?
Options: ( [Yes] | [No] | [C]ancel)
Default: [Y]
```

You should consider adding a call to this shell script in your session startup file (**.login** or **.profile** on most UNIX systems).

Preparing to Install Dynamic 4GL on Windows NT

Before you install Dynamic 4GL on Windows NT, check that your system meets the following requirements.

If you are also planning to install an Informix database server on Windows NT, refer to the *Administrator's Guide for Informix Dynamic Server* for installation and configuration information.



Tip: If you are planning to install Dynamic 4GL, you should install the Dynamic 4GL Windows Client first. For more information on how to install the Windows Client, refer to [“Installing the Windows Client”](#) on page 12-5.

C-Compiler Requirement

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

Informix Database Server Requirement

You must have at least one version of the ESQL/C development package installed to create your own P-code runner.

The Dynamic 4GL development packages for Windows NT install two default runners. One runner does not include an Informix database interface and is called **fglnodb**. One of five others is also installed, depending on the version of the Informix database 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 that is compatible with your Informix database.



Important: Use ESQL/C Version 7.20.TE1 or higher because Version 7.20.TD1 might cause system instability on Windows NT 4.0. You can also download the latest Informix Client SDK without charge from the Informix web site. For more information, go to www.informix.com and choose **Products** → **Connectivity and Gateways** → **Free Download**.

TCP/IP Requirement

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



Important: Only the Microsoft TCP/IP stack is supported. Problems might occur with other TCP/IP stacks.

Hardware Prerequisite

A network card is required.



Warning: Changing the network card disables the license information.

Recommended Windows Client Prerequisite

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

Installing Dynamic 4GL on Windows NT

This section describes how to install Dynamic 4GL on Windows NT. For instructions on how to install the Windows client, see [“Installing the Windows Client” on page 12-5](#).

Dynamic 4GL Installation

An installation program is provided. To perform a manual installation, see [“Post-Installation Tasks” on page 2-21](#).

To install the development and runtime packages on Windows NT

1. Insert your CD.

This procedure assumes it is on the D drive.

2. Execute the installation program.

For the runtime package enter:

```
D:\os\nt\runtime\setup.exe
```

For the development package enter:

```
D:\os\nt\development\setup.exe
```

The Installation window appears.

3. Click **Continue**.

The installation program will look for the Informix database and ESQL/C version and will localize these products. You can choose among the following three modes of installation:

- Automatic search of an existing Informix version.** This option searches the database registry for all needed information (database server or INFORMIX-CLI).
- Specify the Informix directory.** This option 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 computer (database server or INFORMIX-CLI).

Normally, you will use the automatic search.

4. Click **Accept** to continue.

Click **Refuse** to go back if the file path to the Informix product is not correct.

If any Informix product is found, the installation program displays a dialog box that says the installation program will install a runner for the Informix database and a runner for the non-database application.

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

5. 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.

The INFORMIX License Manager Program dialog box allows you to install or remove a license, as [Figure 2-1](#) shows.

Figure 2-1
INFORMIX License
Manager Program
Dialog Box



Tip: You enter license information only once. If you need to reinstall the product, you do not need to enter the license information again unless you removed the old %FGLDIR% directory structure.

The serial number and serial number key are provided with the Dynamic 4GL media. The installation number is generated for you. To access the installation key, go 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: twelve *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).

Every field must be completed, with the following exceptions:

- The **Installation Number** field is automatically computed and therefore you do not need to complete it. 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 enable you to check that the corresponding serial number has been entered correctly.



***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.*

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.

Configuring Dynamic 4GL for Windows NT

After installing Dynamic 4GL, you can configure the product. The following steps show how to setup Dynamic 4GL for a database server. For more information on how to configure the database server, see “[Configuration Files](#)” on [page 12-37](#).

1. 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**.

2. 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
to
SET FGLPROFILE=C:\usr\FGL2C\ETC\magellan.prf
```

3. Save the **magellan.bat** file.
4. 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
```

5. Compile this program with the following command:

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

6. Run the program with the following command:

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

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

```
Status:0
```

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

Connecting to a Windows NT Database Server

You need an rlogin service to connect the Informix database server with a Dynamic 4GL client. Windows NT does not include an rlogin service. However, you will find several rlogin solutions for Windows NT on the Dynamic 4GL CD. For the directions for installing and configuring the Ataman remote login service software, refer to [“Installing and Configuring the Ataman Remote Login Service”](#) on page 2-22.

The Ataman software supplied with Dynamic 4GL is a demonstration version that you can use for 30 days. If you are interested in using the software after 30 days, you will be required to purchase a licensed version. For more information, go to www.ataman.com.

Dynamic 4GL Directories

This section describes the directories created during the installation process. All the files that the compiler installs are under the directory specified during the installation and are referenced by the environment variable **FGLDIR**. These directories contain the following information.

Name	Description
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

(1 of 2)

Name	Description
lock	The data files created by clients running compiled applications (Removing this directory while applications are running leads to a failure of all the currently running 4GL programs.)
msg	The compiled error and runtime messages handled by the compiler
release	The latest documentation about new features, corrected bugs, and known problems and their workarounds
src	Source files of tools and 4GL libraries (It also contains the readable form of the error messages contained in the msg directory.)
toolbars	The icons you can include in the toolbars of your applications

(2 of 2)

Post-Installation Tasks

If you are performing an automatic installation, the following tasks are done for you. If you are doing a manual installation, you need to complete the following procedures manually before you can use Dynamic 4GL. For more information see [“Post-Installation Tasks” on page B-3](#).

- Find the required libraries: **findlib.sh**
- Create the P-code runner and libraries
- Create the C-code libraries

Installing and Configuring the Ataman Remote Login Service

The following instructions show how to install and configure the Ataman Remote Login Service. A 30-day demonstration of this software is included with Dynamic 4GL.

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

```
C:\usr\ataman atrls install
```
2. Click the **Ataman** icon in the Control Panel. The Ataman TCP Remote Services dialog box appears.
3. Click the **Users** folder and then click **Add User**. The dialog box shown in [Figure 2-2](#) appears.

Add User

User Name:

NT User Name:

NT User Domain:

Home Directory(*) :

Interactive Cmd Processor(*) :

Batch Cmd Processor(*) :

Environment File(*) :

Fields marked with (*) are optional.

Required for Rshd or Rlogind (logon without password) Only

NT Password:

Host Equivalence List:

OK Cancel

Figure 2-2
Add User Dialog Box

4. Click the **Advanced** tab, as [Figure 2-3](#) shows.

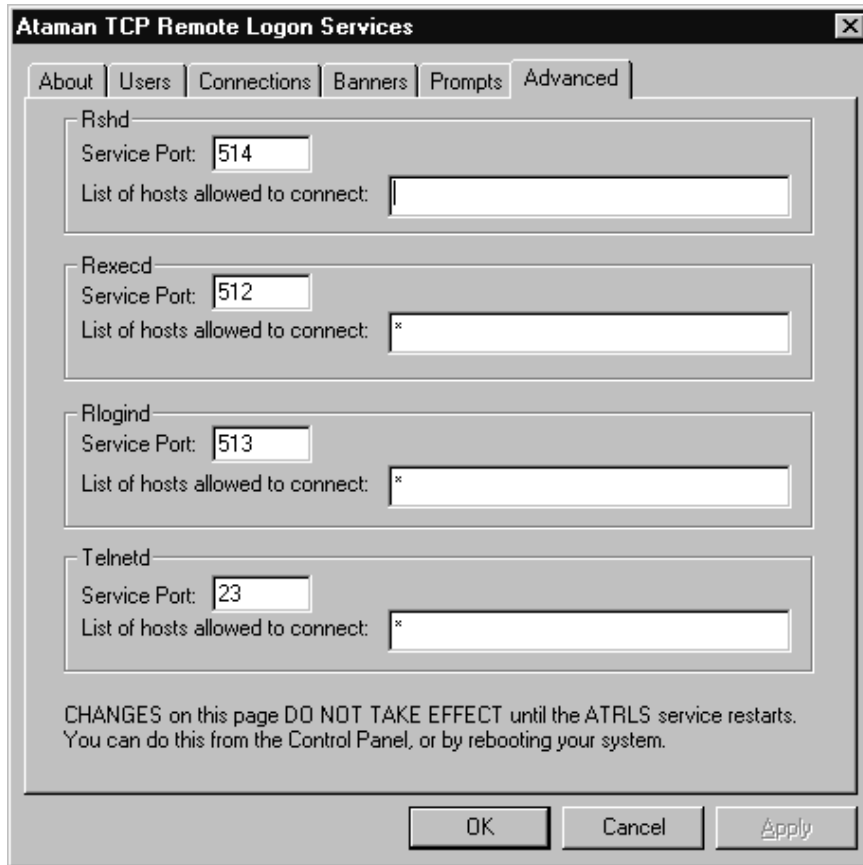


Figure 2-3
Advanced Page

5. Remove the asterisk (*) in the **List of hosts allowed to connect** text box and click **OK**.
6. Start the Ataman remote connection by running the following program at the command prompt:

```
C:\usr\ataman atrls start
```

7. 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 ESQLMF FET_BUFF_SIZE GL_DATE  
GL_DATETIME INFORMIXDIR INFORMIXSERVER  
INFORMIXSQLHOSTS LANG LC_COLLATE LC_CTYPE  
LC_MONETARY LC_NUMERIC LC_TIME DBALSBC  
DBAPICODE DBASCIIIBC DBCENTURY DBCODESET  
DBCONNECT DBCSCONV DBCSOVERRIDE DBCSWIDTH DBFLTMSK  
DBMONEYSCALE DBSS2  
DBSS3
```

8. Change the following 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
```

9. Save these modifications and then create a connection using **Wtk**, as **Figure 2-4** shows.

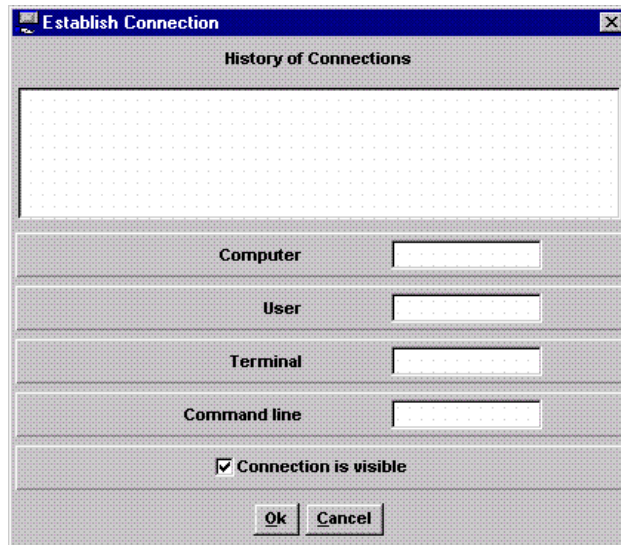


Figure 2-4
Establish Connection
Dialog Box

10. When the connection is made, enter the password.

You will be in the %FGLDIR% directory.

11. Start your environment file **magellan.bat**.

Now you can run **testdbs.42r**. This program indicates that your status is set to 0, as [Figure 2-5](#) shows, which means that the connection was successful.

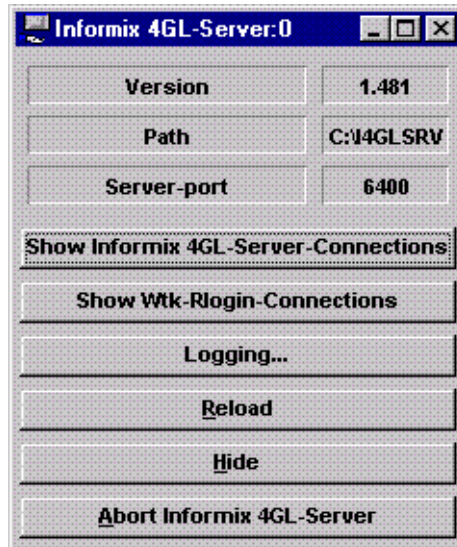


Figure 2-5
4GL-Server
Dialog Box

Basics of Using Dynamic 4GL

In This Chapter	3-3
Setting the Environment Variables	3-3
Compiling a Simple Program	3-4
Writing the Source Code	3-5
Compiling the Source Code	3-6
Compiling to P Code	3-6
Compiling to C Code	3-7
Compiling the Form-Specification File	3-7
Viewing the Dynamic 4GL Application	3-8

In This Chapter

This chapter explains how to set environment variables and compile a simple program in Dynamic 4GL.

Setting the Environment Variables

During installation, a file is created that sets the environment for your configuration. This file is located in the directory where you installed Dynamic 4GL and is named **envcomp**.

The first task is to set up correct environment variables. To do this, execute the **envcomp** script located in the **\$FGLDIR** directory. For instance, if the compiler is installed in the **/usr/fgl2c** directory, type the following Bourne shell commands:

```
$ cd /usr/fgl2c
$ . ./envcomp
```

This script adds the following environment variables.

Variable	Description
FGLDIR	The directory specified during the installation for Dynamic 4GL
INFORMIXDIR	The Informix root directory specified during the installation of Dynamic 4GL
FGLDBS	Tells the compiler which version of the Informix interface you installed on your computer
FGLCC	Name of the C compiler you want to use

(1 of 2)

Variable	Description
FGLLIBSQL	List of the required Informix SQL libraries
FGLLIBSYS	List of the required system libraries
FGLGUI	Used only at runtime to specify if the program should be executed in graphical mode or in ASCII mode
PATH	Search path for required files and components. The script adds the \$FGLDIR/bin directory to your PATH
FGLSHELL	External shell for linking system tools (CC, GCC), or Informix tools (ESQL, C4GL) to compile a P-code runner
LD_LIBRARY_PATH	Location of shared libraries.

(2 of 2)



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

This file is a Bourne shell script. If you are using a UNIX 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.

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:

- Writing the 4GL source code
- Compiling to P code or C code

After you compile the program, you compile the form-specification file.

Writing the Source Code

The first step is to write the 4GL source code for your application. The following sample program is made of two 4GL modules and one form-specification file.

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

```

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**:

```

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**:

```

DATABASE FORMONLY
SCREEN {

    Icon [bt1]      Title[01]

                        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";

```

All the strings between the double quotes are case sensitive.

Compiling the Source Code

The next step is to compile this 4GL source code. You can compile to either P code or C code.

Compiling to P Code

P code has many advantages over C code. The main advantages of using P code are:

- you compile it once and then you can run the same compiled modules on every computer on which a Dynamic 4GL runtime package is installed.
- all the new 4GL features are implemented in P code only.

To compile **ex1-1** to P code, change to the directory where you created the **ex1-1.4gl** and **ex1-1.per** files. Check that your environment variable is correctly set:

```
$ echo $FGLDIR
```

This statement should return the directory where you installed Dynamic 4GL. 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 that contains calls to the functions included in the **.42m** modules. It is absolutely necessary to keep these modules accessible at runtime.

Compiling to C Code

You can compile the sample program to C code. However, C code is only available for UNIX platforms. To compile to C code, use **fgl2c** instead of **fgl2p**.

In addition, you need the following: a C compiler, a linker, and two environment variables (**FGLLIBSYS** and **FGLLIBSQL**). These two environment 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
```

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-Specification File

Form files are compiled with the **fglform** compiler. Compiled forms can be used by both P-code and C-code programs. To compile the form-specification file **ex1-1.per**, type the following:

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

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

Viewing the Dynamic 4GL Application

The Dynamic 4GL application can be viewed using a client interface. The following illustration shows how the application will look when viewed using the X11 Client. For more information about installing the X11 Client, and viewing the application, refer to [Chapter 13, “Using the X11 Client.”](#)

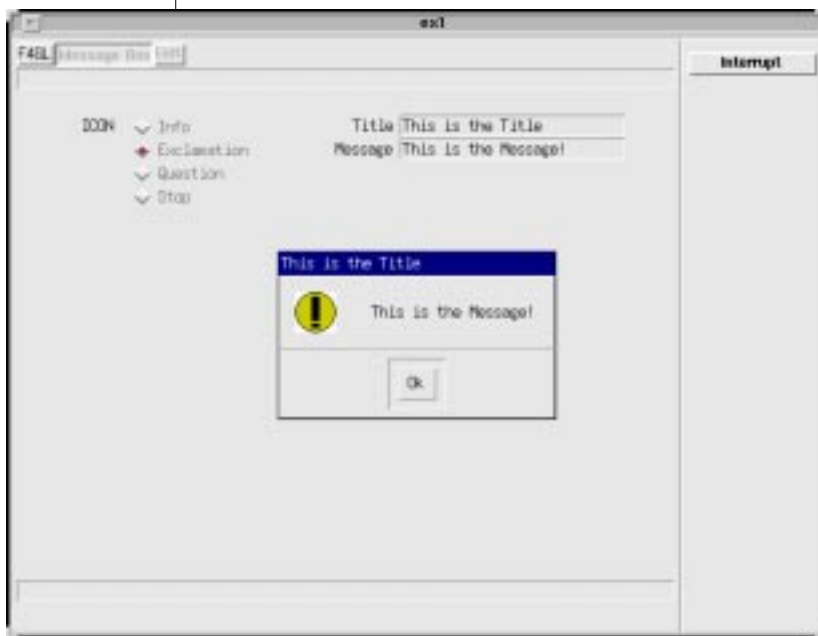


Figure 3-1
*Application Displayed
Using the X11 Client*

Using the Dynamic 4GL Compiler

In This Chapter	4-3
Setting Environment Variables for the Compiler	4-3
Compiling Form-Specification Files and Help Message Files	4-4
Compiling Form-Specification Files	4-4
Compiling Help Message Files	4-5
Generating a Database Schema File	4-5
Compiling to P Code	4-6
Overview of a P-Code Example	4-6
Compiling Source Files to Linkable Modules	4-7
Linking Modules Together to Create P Code	4-7
Using C Functions in 4GL Applications	4-8
Compatibility Problems with C	4-8
Linking C Functions with the P-Code Runner	4-8
Examples	4-12
Compiling to C Code	4-19
Overview of a C-Code Example	4-19
Compiling Source Files to Linkable Modules	4-20
Linking Modules to Create C-Code Libraries	4-21
Using C Functions in 4GL Applications	4-21
Linking C Functions for Use in C-Code Compilations	4-21
Compilation Tools	4-26
Main Compilation Tools	4-27
Other Compilation Tools	4-27
Configuration Tools	4-28
Miscellaneous Programs and Scripts	4-28

In This Chapter

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

This chapter covers the following topics:

- Setting environment variables for the compiler
- Compiling 4GL form specification files and help message files
- Generating a database schema file
- Compiling to P code
- Compiling to C code
- Creating a P-code runner with **fglmkrun**
- Compilation tools

Setting Environment Variables for the Compiler

The first task is to set up correct environment variables. To do this, execute the **envcomp** script located in the **\$FGLDIR** directory. This shell script was created during installation and it sets up the main environment variables required to use Dynamic 4GL.

For instance, if the compiler is installed in the **/usr/fgl2c** directory, type the following Bourne shell commands:

```
$ cd /usr/fgl2c
$ . ./envcomp
```

Important: You should include a call to this shell script in your session startup file (*.login* or *.profile* on most UNIX platforms).



You can control the behavior of the Dynamic 4GL compiler with configuration files. For more information, see [Chapter 8, “Configuring the Dynamic 4GL Compiler.”](#)

Compiling Form-Specification Files and Help Message Files

This section describes how to compile the form-specification files and help message files that your 4GL applications use.

Compiling Form-Specification Files

You need to recompile 4GL forms into a Dynamic 4GL format. The compiled form files can be used with applications compiled to either P code or C code.

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

```
$ fglform formname{.per}
```

This command compiles the form-specification file named **formname.per** into **formname.42f**. The **.per** extension is not mandatory on the command line.



Important: *Because you might need to recompile forms on a computer other than your development computer, the **fglform** compiler is installed with the Dynamic 4GL runtime package.*

Compiling Help Message Files

You need to recompile the 4GL help message files into a Dynamic 4GL format. The help message compiler, named **fglmsg**, is similar to the Informix **mkmessage** compiler. The compilation syntax is as follows:

```
$ fglmsg input_file output_file
```

The following command decompiles the file and the output is written to the standard output:

```
$ fglmsg -r compiled_file
```

Generating a Database Schema File

You need to generate a database schema file. Run the following command in a directory and environment where you have access to your database (for instance, 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. If needed, two other files are generated: **database_name.att** and **database_name.val**. These files manage the **syscolatt** and **syscolval** tables (respectively).

The environment variable **FGLDBPATH** must include the path to the directory that contains this database schema file. This environment variable allows you to compile 4GL programs that reference the database.

Important: While no changes are required to the database before using Dynamic 4GL, a schema file must be generated each time that the database structure changes.



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.

P code is hardware-independent pseudo-executable code. The same P code can be executed on any operating system 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.

Overview of a P-Code Example

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

```

MAIN
DISPLAY "Hello World"
END MAIN
    
```

Before executing this program, you need to compile the code and then link the needed modules and the P-code runner. [Figure 4-1](#) shows the complete compilation schema.

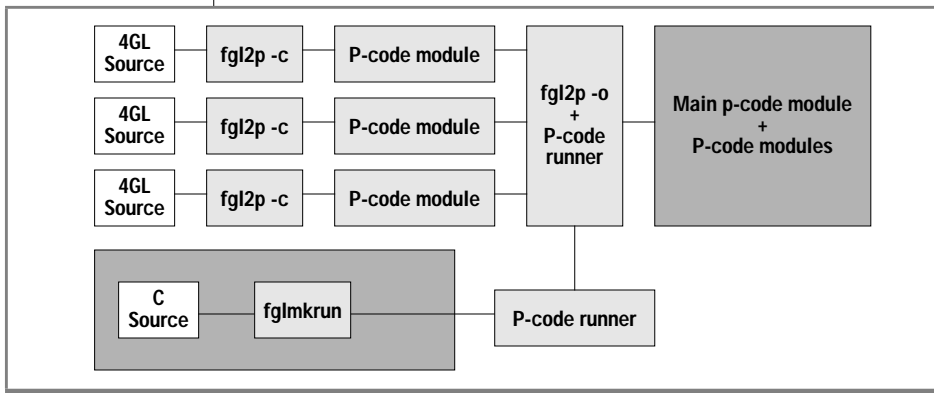


Figure 4-1
P-Code
Compilation
Schema

The name of the Dynamic 4GL P-code compiler is **fgl2p**. This tool compiles 4GL source code into P-code executables or libraries.

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:

```
$ fg12p 4gl_source_code.4gl
```

For example:

```
$ fg12p example.4gl
```

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

Linking Modules Together to Create P Code

The following syntax links the compiled **.42m** modules together to create an executable or library. This link also checks for C functions included in the runner (see “[Linking C Functions with the P-Code Runner](#)” on page 4-8) that the **FGLRUN** environment variable specified.

```
$ fg12p -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**:

```
$ fg12p -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.

At runtime, all modules linked together must be located in a directory specified by the **FGLLDPATH** environment variable.

The **.42m** modules are linked together into the **.42r** hash table that contains cross-references to all functions and variables used in the 4GL application. Thus, all unresolved or faulty references (for instance, missing functions, or 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 that contain the MAIN section are loaded into memory. All other **.42m** modules are loaded when needed. Every module and all library modules appear only once in the application. This can lead to a significant reduction in the size of the P-code modules constructing the application.

Using C Functions in 4GL Applications

This section describes a strategy for using C functions in your application.

Compatibility Problems with C

Using C functions in your 4GL applications can cause problems when you port the application to a platform other than the one used to develop it. For example, you can expect problems when porting an application from UNIX to Windows NT and vice versa. Problems 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 that allow 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 Form Extensions to 4GL.”](#)

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, you 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** shell script allows you to generate a specific runner with C function. For more information, you can view the output of **fglmkrun**. You can also use different flags and options with **fglmkrun**.

Viewing Sample fglmkrun Output

You can see what was done during **fglmkrun** execution. The output displays the following information:

- The name and location of the created P-code runner
- The compiler/linker used by the script to build the P-code runner (for example, esql, gcc, cc, c4gl, and so on)
- The list of the additional flags and libraries added to the command line
- The current value of **\$INFORMIXDIR**
- The version of Informix database interface for which the runner is created:
 - ix410 for 4.10 Informix interfaces
 - ix501 for Informix interfaces from 5.01 to 6.X
 - ix711 for Informix interfaces 7.X
 - ix914 for Informix database interfaces 9.13 or greater
 - ixgen for all Informix database interfaces but requires the compilation of an ESQ/C source file (**\$FGLDIR/src/esql_gen.ec**). In this case the only compiler that can be used for creating the runner is esql.
- The memory mapping used is the system built-in or an emulation. For information on memory mapping see [“Memory Mapping Settings” on page 8-37](#).
- The list of all files added to the compiler
In most instances, these are C-function source files called from the 4GL source code.

To view the P-code runner output, execute **fglmkrun**. The following information appears:

```
The runner was successfully linked with the following options:
Runner name           : /usr/fgl2c/bin/fglrun
Language Support Library : ASCII
Compiler              : esql
Additional flags/libs  : None
Informix dir          : /informix
Database interface     : ix914 (/usr/fgl2c/lib/libix914.a)
Memory mapping        : System built in
User extensions       : None
```



Important: Be sure that the environment is correct before executing *fglmkrun*. If necessary, run the *findlib.sh* shell script and use the resulting shell script.

Building a Statically Linked Runner

To build a statically linked P-code runner named **myrun** using the Client SDK, Version 2.10, and a C-function file name **file.c** (assuming the prototype of these functions are defined in the file **\$FGLDIR/lib/fglExt.c**), execute **fglmkrun** with the following command:

```
$ fglmkrun -d ix914 -add -static $FGLDIR/lib/fglExt.c file.c -o myrun
```

The following output appears:

```
The runner was successfully linked with the following options:
Runner name           : myrun
Language support library: ASCII
Compiler              : esql
Additional flags/libs  : -static
Informix dir          : /ix/informix.csdk
Database interface     : ix914 (/work/pl/fgl2c/lib/libix914.a)
Memory mapping        : System built in
User extensions       : Yes
/work/pl/fgl2c/lib/fglExt.c
file.c
```

Details About fglmkrun

You can use the following options when executing **fglmkrun**:

Syntax	<code>fglmkrun options [ext]</code>
Options	
-V	Display fglmkrun version information.
-h	Display help message with a list of fglmkrun options.
-vb	Verbose mode displays the compilation line used and fglmkrun output.
-o <i>name</i>	Output to <i>name</i> , default=fglrun. This symbolic link is to the default runner created either in the \$FGLDIR/bin/gls directory if the flag -gls is set and in \$FGLDIR/bin/ascii if it is not set.
-d <i>dbver</i>	Database interface version, default=ix410 <ul style="list-style-type: none"> ■ ix410 : Informix 4.10 ■ ix501 : Informix 5.01 ■ ix711 : Informix 7.11 ■ ix914 : Informix 9.1x and higher ■ ixgen : Generic Informix 9.1x and higher
-sh <i>prog</i>	External shell for linking (esql, c4gl, and so on). The fglmkrun script can use the system tools (cc, gcc) or Informix tools (esql, c4gl) to compile a P-code runner. For system tools, the required Informix libraries and system libraries have to be set with the findlib.sh script. For Informix tools, the Informix tools automatically find the libraries. Default value is CC.
-add " <i>other</i> "	Add other system libraries or flags to link. The -add flag sends the specified parameter to the compiler building the runner. The argument will not be interpreted by the fglmkrun script.
<i>ext</i>	List of user extension modules. This can be anything else that is to be compiled and linked with the runner such as libraries, C files, object files, and so forth.
-gls	Set this option is you want to use the GLS language support library. To create a runner with GLS support, you need the Client SDK 2.01 (or later) database interface. If the -gls flag is not specified, a runner handling only the ASCII charset is created.

For a list of **fglmkrun** error messages, see “[fglmkrun Errors](#)” on page -34 of Appendix C.

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

Examples

The following example shows the standard extension file **\$FGLDIR/lib/fglExt.c**:

```
#include "f2c/fglExt.h"
UsrData usrData[]={
  { 0, 0 }
};
UsrFunction usrFunctions[]={
  {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:

```
#include "f2c/fglExt.h"
#include "f2c/r_c.h"
int my_func1_cname(int nargs);
int my_func2_cname(int nargs);
UsrFunction usrFunctions[]={
  { "my_func1_4glname", my_func1_cname, my_func1_nbparam, my_func1_nbret },
  { "my_func2_4glname", my_func2_cname, my_func2_nbparam, my_func2_nbret },
  { 0,0,0,0 }
};
```


The following table describes the elements in this example.

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

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

First create your C file (**example.c**):

```
#include <stdio.h>
int fnccl(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. Then edit **fglExt.c** and update it with the following definitions:

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

Now 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, system libraries, Dynamic 4GL libraries, and the file **example.o**) named **newrunner**. This runner is for 4.x Informix databases.

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:

```
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** environment variable to determine which runner to link with. If you do not set the **FGLRUN** environment variable before you link 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
```

Calling 4GL from C

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

```
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:

```
#include <stdio.h>
#include "f2c/fglExt.h"
int fnccl(int n)
{
  fCall( "fnc2",0);
  return 0;
}
```

Compile these statements using the following command:

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

Now update file **fglMyExt.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:

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

Then update the 4GL example:

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

Compile it with **fgl2p**:

```
$ 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:

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

Modifying 4GL Global Variables From C Functions

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 that contain your C functions.

Furthermore, every variable must be defined as external to the C module with its corresponding type. Use the following syntax:

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

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

The following example shows a file that contains the C functions called from 4GL:

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

Compile the C file with the following command:

```
$ 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:

```
GLOB_type(varname[, varlength]);
```

The following table describes the elements of this command.

Element	Description
<i>type</i>	The type of the variable
<i>varname</i>	Name of the variable as defined in 4GL
<i>varlength</i>	Length of the variable as defined in 4GL (only for CHAR variables)

The code is:

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

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

Create the new runner:

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

The following table shows the supported data types.

CHAR	GLOB_CHAR GLOB_VARCHAR
SMALLINT	GLOB_SMALLINT
INTEGER	GLOB_INT
SMALLFLOAT	GLOB_SMALLFLOAT
FLOAT	GLOB_FLOAT

(1 of 2)

DECIMAL	GLOB_DECIMAL
MONEY	GLOB_MONEY
DATE	GLOB_DATE

(2 of 2)

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

Global RECORD and ARRAY statements are not allowed.

Adapt your 4GL example as follows:

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

Compile it with **fgl2p** and run it with the new runner:

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

Building a Runner on SCO Systems

With SCO systems, the use of **fglmkrun** during a manual installation causes the following error message:

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

This problem arises because of differences between the various versions of the SCO libraries. The solution to this problem is to create a file named **fileno.c** that contains the following lines:

```
#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 fgldrun fileno.c $FGLDIR/lib/fglExt.c
```

Compiling to C Code

While Dynamic 4GL allows you to compile 4GL programs to C code, consider compiling to P code rather than C code. With C-code compilation, you must recompile the whole program whenever you change the execution platform, whereas with P code, you only need to rebuild your runner. In addition, P code does not execute significantly slower than C code.

Overview of a C-Code Example

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

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

Before you execute this program, you first need to compile it and then link all the needed modules. [Figure 4-2](#) shows the complete compilation schema.

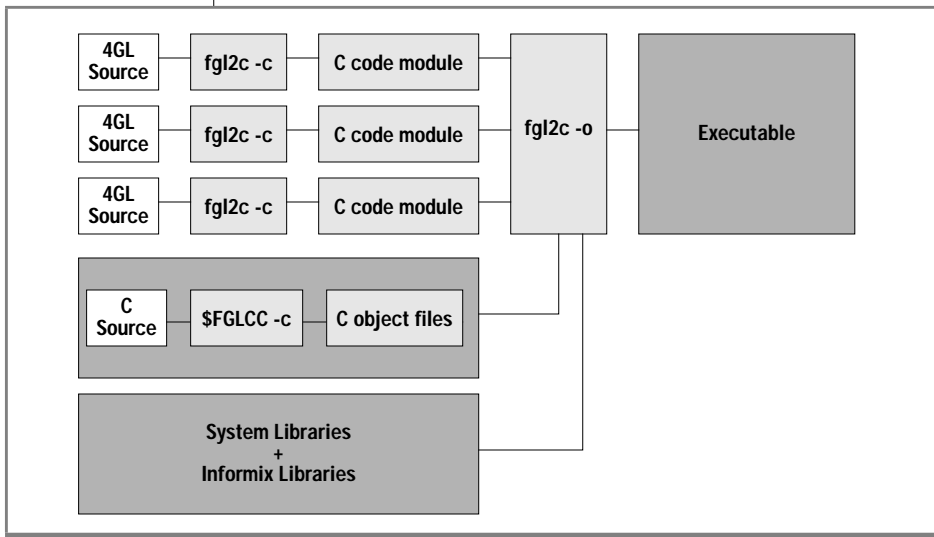


Figure 4-2
C-Code
Compilation
Schema

The name of the Dynamic 4GL C-code compiler is **fgl2c**. This tool compiles the 4GL source code into C-code executables or into libraries.

Compiling Source Files to Linkable Modules

The following conventions are used for the filename extensions:

- **.4gl** for the source-code files
- **.42o** for the compiled modules
- **.42e** for the file resulting from the linking of compiled modules, system libraries, and P-code libraries

The following syntax is the first step of the compilation, which compiles 4GL source code into linkable modules:

```
$ fgl2c -c 4gl_source_code.4gl
```

For example:

```
$ fgl2c -c example.4gl
```


This line compiles the 4GL source-code file **example.4gl** to the module **example.42o**.

Linking Modules to Create C-Code Libraries

The following syntax links 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

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

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. You will also have to include the **f2c/r_c.h** header file at the beginning of your C files. Use the following syntax to call the macro:

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

The following table describes the elements of this command.

Element	Description
<i>C_function_name</i>	Name of the C function
<i>parameters_number</i>	Number of parameters transmitted to the function
<i>returned_values_number</i>	Number of values returned by the function

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 4GL compilers, use conditional compiling, as in the following example:

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

The following example shows a 4GL application that calls a C function named **mainc**. The 4GL source-code file is named **exCCCode.4gl** and contains the following code:

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

The C source file is named **exc.c** and contains the following code:

```
#ifndef Informix
#include "f2c/r_c.h"/* This is the Informix header file defining the CFACE
macro */
#endif
#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 following 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** macro in your C functions. This function is also defined in the **f2c/r_c.h** Dynamic 4GL header file. The syntax is as follows:

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

The following table describes the elements of this command.

Element	Description
<i>4GL_function_name</i>	Name of the 4GL function
<i>parameters_number</i>	Number of parameters transmitted to the function
<i>returned_values_number</i>	Number of values returned by the function

The following example shows 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.

The first 4GL module is **exCCode.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**:

```
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**:

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

The C statements **popquote**, **pushquote**, **pop[...]** and **push[...]** are working exactly as with INFORMIX-4GL compilers.

Next, compile these three files:

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

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

```
$ fgl2c -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. The syntax of the **CNAME** macro is:

```
#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.

The first new 4GL module is **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 fnccl()
IF word = "How are you?" THEN
    DISPLAY "Very fine and you?" AT 10, 1
END IF
END FUNCTION
```

The new C function is **exc.c**:

```
#ifndef Informix
#include "f2c/r_c.h"
#define word CNAME(word) /* here is the variable declaration in the C file
*/
#endif
#include <stdio.h>
extern char word[13]; /*here is the prototype of the variable in the C
file */
int mainc(int n)
{
printf("%s\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
```

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.

Filename	Description
fgl2p	Script to compile applications to P code
fgl2c	Script to compile applications to C code
fglcomp	Main compiler program
fgllink	Main linking program
fglform	Tool for compiling form specification files (.per)
fglschema	Script to create a schema of your databases used by the 4GL compiler at compile time
fglmkrun	Script to create a new P-code runner
fglnodb	The default P-code runner without any database interfaces
fglrun	The P-code runner created during the installation process, including your Informix interface

Other Compilation Tools

Use the following scripts to create archives and locate libraries.

Filename	Description
ar42o	Script to create archive files from .42o object files
findlib.sh	Script to find all the libraries needed on your system to create P-code runners or C-code executables

Configuration Tools

The following table lists tools that aid in configuration.

Filename	Description
fglmkmsg	Tool to create the runtime error message libraries
licencef4gl	Script to install a license
confdesi	Script to start the configuration program for the X11 interface
fglfontsel	Font selection tool for X11 interfaces (P-code version)
fglfontsel.42e	Font selection tool for X11 interfaces (C-code version)

Miscellaneous Programs and Scripts

This table lists other helpful tools.

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

Using Non-Graphical Extensions to 4GL

In This Chapter	5-3
Channel Extensions	5-3
Initializing Channel Extensions	5-4
Opening a File	5-4
Opening a Pipe	5-5
Setting the Default Separator	5-6
Reading Data from an Opened Channel	5-6
Writing Data to a Pipe or Stream	5-7
Closing the Channel.	5-8
Channel Error Codes	5-8
Sharing Information Using DDE	5-8
Supported Windows Applications	5-9
Using DDE Extensions	5-9
Transmitting Values to a Windows Program	5-11
Getting Values from a Windows Program.	5-12
Closing a DDE Connection.	5-13
Closing all DDE Connections	5-13
Extending the DISPLAY ARRAY Statement	5-14
Returning Key Code Values	5-15
Returning Key Codes from P Code	5-16
Returning Key Codes from C Functions	5-18
Creating a Custom Character Filter	5-18

Starting a UNIX Emulator	5-19
Starting Windows Applications	5-20
Using Input Statement Functions	5-21
Returning a Value if a Field has been Modified	5-21
Returning the Name of a Field.	5-23
Returning the Value of a Field	5-23
Setting the Value in a Field	5-23
Displaying a Row at a Given Line in a Screen Array	5-24
Returning the Position of the Cursor	5-27
Setting the Cursor Position	5-28
Terminating Applications.	5-29
New Language Features	5-29
Enhanced SQL Syntax Support	5-30
Support For Embedded SQL 7.3 Syntax	5-30
Support for Preparable SQL Statements	5-31
Syntax for Expansion of Abbreviated Year Values	5-34
Legacy Support for DBCENTURY	5-35
New CENTURY Field Attribute	5-36
New CENTURY Display Attribute in PROMPT Statements	5-37
Enhanced Syntax for Screen Array Management	5-38
Data Editing in Screen Arrays	5-38
New CURRENT ROW DISPLAY Attribute	5-41
New COUNT Attribute.	5-43
New MAXCOUNT Attribute.	5-43
New FGL_SCR_SIZE() Built-In Function	5-44
Dynamic Configuration of Report Output.	5-46
New Built-In Operators	5-48
String Concatenation Operator	5-48
Synonym for the Equality (=) Relational Operator	5-49
New Syntax to Hide the Comment Line	5-50
Editing Multibyte Data in 4GL Forms	5-51
New Conditional Comments	5-53

In This Chapter

This chapter describes nongraphical extensions (extensions that do not affect the database interface) that can be used to enhance Dynamic 4GL applications. This chapter includes the following sections:

- Channel extensions
- Sharing information using Dynamic Data Exchange (DDE)
- Extending the DISPLAY ARRAY command
- Returning key code values
- Starting a UNIX emulator
- Starting Windows applications
- Using input statement functions
- Terminating applications
- New language features

Channel Extensions

Channel extensions provide access to the system, the files, and the processes, without using the RUN statement. With channel functions, your application requires fewer resources (than the RUN statement) and allows you to communicate through pipes with other applications.

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

In the section [“Sharing Information Using DDE” on page 5-8](#), you will find examples that illustrate the use of the DDE functions.

Initializing Channel Extensions

The following statement tells the compiler that channel extensions will be used during the execution of the 4GL program:

Syntax USE channel

Returns None

This statement must be located before the MAIN clause in the source code. For example:

```
USE channel
MAIN
...
END MAIN
```

Opening a File

The following function opens the file specified by *filename* and prepares the file for reading or writing, as specified by *oflag*:

Syntax channel::open_file(*handle*, *filename*, *oflag*)

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)

Returns None

The filename is assigned to the handle that will be called for the different operations on the opened channel. For example:

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

Opening a Pipe

The following function opens the pipe specified by *command* and prepares the pipe for reading or writing, as specified by *oflag*:

Syntax	channel::open_pipe(<i>pipe_handle</i> , <i>command</i> , <i>oflag</i>)	
<i>pipe_handle</i>	CHAR(<i>xx</i>)	Unique identifier for the specified command
<i>command</i>	CHAR(<i>xx</i>)	Name of the command you want to execute
<i>oflag</i>	CHAR(1)	<ul style="list-style-type: none"> 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	

The command is assigned to the handle called for the different operations on the opened channel. For example:

```
CALL channel::open_pipe("pipe", "ls -l", "r")
```

Setting the Default Separator

The following function allows you to change the delimiter of each opened channel defined by its handle within a 4GL program:

Syntax	<code>CALL channel::set_delimiter(<i>handle</i>, <i>delimiter</i>)</code>
<i>handle</i>	CHAR(<i>xx</i>) Unique identifier for open channel
<i>delimiter</i>	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 **DBDELIMITER** environment variable. The default value is the | (pipe) character. If `delimiter=""` (empty string), no delimiter is used. For example:

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

Reading Data from an Opened Channel

The following function reads data from the stream specified by the handle and stores the data in a buffer.

Syntax	<code>channel::read(<i>handle</i>, <i>buffer-list</i>)</code>
<i>handle</i>	CHAR(<i>xx</i>) Unique identifier for open channel
<i>buffer-list</i>	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 <i>handle</i> ; FALSE if an error occurs

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 computers and a general protection fault on Windows systems.



The following examples show this function. The first example shows a **read** function return value in a variable buffer.

```
DEFINE buffer CHAR(128)
CALL channel::read("pipe_handle", buffer) RETURNING ret
```

The second example shows a **read** function returning data in a simple array:

```
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
```

The third example shows a **read** function returning data in a record:

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

Writing Data to a Pipe or Stream

The following function writes data from a stored buffer to a 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

The storage buffer can be a single variable, a simple array, a record, or a string between double quotes (“”). For example:

```
CALL channel::write("handle", "hello world")
```

Closing the Channel

The following function closes the channel specified by *handle*:

Syntax	<code>channel::close(<i>handle</i>)</code>
<i>handle</i>	CHAR(xx) Unique identifier for open channel
Returns	None

For example, assume *handle* is called **handle1**:

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

Channel Error Codes

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:

- 6340: Cannot open file.
- 6341: Unsupported mode for 'open file'.
- 6342: Cannot open pipe.
- 6343: Unsupported mode for 'open pipe'.
- 6344: Cannot write to unopened file or pipe.
- 6345: Channel write error.
- 6346: Cannot read from unopened file or pipe.

Sharing Information Using DDE

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 Client.

You will find an example that illustrates DDE and the channel features in the section, "[Creating a Custom Character Filter](#)" on page 5-18.

Supported Windows Applications

Dynamic 4GL supports data exchange for the following Windows applications:

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



Important: The DDE extension might not run with the latest Microsoft Office versions (such as Office 97). These applications do not fully support DDE. For more information, refer to your Microsoft documentation.

Using DDE Extensions

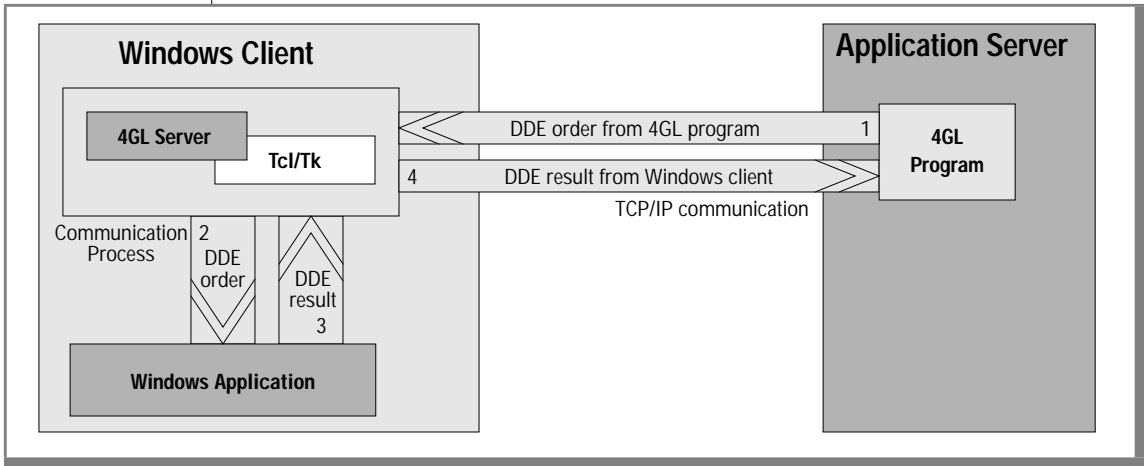
The DDE 4GL process is a four-part process, as follows:

1. The 4GL application sends to the Windows Client (4GL Server) the DDE order using the TCP/IP channel.
2. The Windows Client executes the DDE order using the Tcl/Tk functions and sends the data to the Windows application through the DDE communication process.
3. The Windows application executes the command and sends the result, which can be data or an error code, to the Windows Client.

- The Windows Client sends the result to the 4GL application using the TCP/IP channel.

Figure 5-1 illustrates this process.

Figure 5-1
DDE 4GL Process



To start a Windows application on the client side, use the **winexec** function.

Opening a DDE Connection

The following function opens a DDE connection:

Syntax `DDEConnect (progrname , docname)`

progrname 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")
```

Executing a Program Command Using DDE

The following function executes a command in the specified program using the DDE channel:

Syntax DDEExecute(*progrname*, *docname*, *command*)

progrname CHAR(128) Program name

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

command CHAR(2048) Command 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 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
```

Transmitting Values to a Windows Program

The following function sends data to the specified program and document using the DDE channel:

Syntax CALL DDEPoke(*progrname*, *docname*, *cells*, *values*)

progrname CHAR(128) Program name

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

<i>cells</i>	CHAR(128)	Working items
<i>values</i>	CHAR(128)	Data sent to the <i>progrname</i>
Returns	TRUE if the values have been successfully transmitted; FALSE if an error occurs (The error can be seen using the function DDEGeterror .)	

For example:

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

Getting Values from a Windows Program

The following function gets values from the specified program and stores it in a variable:

Syntax `CALL DDEPeek(progrname, docname, cells)`

progrname 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.)

Each value retrieved by the function is separated by the tabulation character. The newline character is changed to the ASCII 13 character. For example:

```
CALL DDEPeek("EXCEL", "Document1", "R1C1:R2C2") RETURNING ret
```

Closing a DDE Connection

The following function loses the specified DDE channel represented by its unique identifier:

Syntax	<code>CALL DDEFinish(<i>progrname</i>, <i>docname</i>)</code>
<i>progrname</i>	CHAR(128) Program name
<i>docname</i>	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.)

For example:

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

Closing all DDE Connections

The following function closes all DDE connections, as well as the program sending or receiving data on the DDE channels:

Syntax	<code>DDEFinishAll()</code>
Returns	TRUE if all DDE channels have been closed; FALSE if an error occurs (The error can be seen using the DDEGeterror function.)

For example:

```
CALL DDEFinshAll() RETURNING ret
```

Managing DDE Error Messages

The following function retrieves the last error on the DDE channel:

Syntax DDEGeterror()

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

For example:

```
CALL DDEGeterror() RETURNING mess
```

Extending the DISPLAY ARRAY Statement

The following statements extend the DISPLAY ARRAY statement:

```
- 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.



The following example shows 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 fgl_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
.
.
.

```

Returning Key Code Values

You can return a key code value after pressing a keystroke. For P code, the function is **fgl_getkey**. For C code, the function is **uiInkey**.

Returning Key Codes from P Code

The following function waits for a keystroke and returns the key code of a pressed key:

Syntax `fgl_getkey()`

Returns Value of the keystroke

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

```

MAIN
DEFINE key INTEGER

--#CALL fgl_init4js()
--#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 T, you receive the message that [Figure 5-2](#) shows.



Figure 5-2
fgl_winmessage
Window

This function can be used in association with the **fgl_keyval()** function of 4GL. The following table shows the values returned by the **fgl_getkey** function.

Value Returned	Meaning
0 through 255	A single character from the ISO8859-1 character set. This does not apply if you are using a GLS locale with another character set. For more information, see the <i>INFORMIX-4GL Reference</i> .
3000 through 3063	Function keys F1 through F64.
2000	KEY_UP

(1 of 2)

Value Returned	Meaning
2001	KEY_DOWN
2002	KEY_LEFT
2003	KEY_RIGHT
2004	KEY_BACKUP
2005	KEY_NXTSCR
2006	KEY_PRVSCR
2007	KEY_LBSAME
2008	KEY_HELP
2009	KEY_INSCCHAR
2010	KEY_DELCHAR
2011	KEY_INTRPT
2012	KEY_HOME
2013	KEY_END
2014	KEY_INSLINE
2015	KEY_DELLINE
2016	KEY_ACCEPT
2017	KEY_DBINIT
2018	KEY_AUTONEXT (returned whenever an auto-next field is exited, regardless of which key was actually pressed.)
4003	DEL

(2 of 2)

Returning Key Codes from C Functions

In C functions, the equivalent of the function `fgl_getkey()` used to wait for a keystroke is `uiInkey()`, as follows:

Syntax `uiInkey()`

Returns Value of the keystroke

Creating a Custom Character Filter

You can create your own character filter that 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 that contains 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",
# ss,
# 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
"       "
"       "
"       "
```

Starting a UNIX Emulator

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.

Syntax `fgl_system (command)`

In this syntax, *command* is a string or variable that contains the commands to be run. The UNIX terminal will be raised and activated and then lowered later, when the program that needs it finishes.

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:\
```

Starting Windows Applications

The following functions start a Windows program on the computer that runs the Windows Client.

Syntax	<code>WinExec (programe)</code>	Starts a program on the Windows Client without waiting for its end to resume execution of the 4GL program
	<code>WinExecWait (programe)</code>	Starts a program on the Windows Client and waits for its end to resume execution

or

programe	<code>CHAR(256)</code>	Program name with or without its absolute path
-----------------	------------------------	--

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 computer.

Using Input Statement Functions

The following set of functions must be executed inside dialog functions, such as INPUT, INPUT ARRAY, DISPLAY ARRAY, and PROMPT statements. Using these functions outside of a dialog function might create errors at compile time or at runtime.

Returning a Value if a Field has been Modified

The following function is called by AFTER { FIELD | INPUT | CONSTRUCT } and returns a value that indicates whether or not the last field has been modified:

Syntax fgl_buffertouched()

Returns TRUE if the last field has been modified

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")
  --# LET answer = "no"
  --# END IF
  --# END INPUT
  END WHILE
CLOSE WINDOW w1
END MAIN

```

Returning a Value if a Field has been Modified

The first screen displays the form with ***** as the default value, as [Figure 5-3](#) illustrates.



The screenshot shows a window titled "demo" with a form containing the following fields and values:

Code :	*****	Branch :	*****
Name :	*****		
Number :	*****		
Street :	*****		
ZIP :	*****	City :	*****
Country :	*****		
Phone :	*****		
Fax :	*****		

On the right side of the window, there are two buttons: "Accept" and "Interrupt".

Figure 5-3
Default Value
Screen

If you exit the input without updating the row, the dialog box that [Figure 5-4](#) shows appears and informs you that no changes to the row have been made.



Figure 5-4
Exiting Without
Update Screen

If something has changed during the input, the dialog box that [Figure 5-5](#) shows appears.

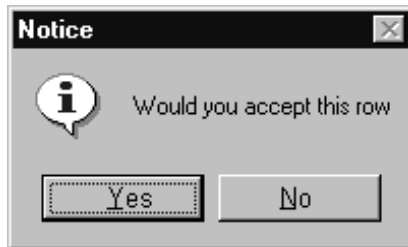


Figure 5-5
Prompt for
Accepting Changes

Returning the Name of a Field

The following function returns the name of the currently prompted field:

Syntax `fgl_dialog_fieldname()` or `dialog::fieldname()`

Returns Name of current field

Returning the Value of a Field

The following function returns the value of the currently prompted field:

Syntax `fgl_dialog_getbuffer()` or `dialog.getbuffer()`

Returns Value of current field

Example See `fgl_dialog_setbuffer()` example below

Setting the Value in a Field

The following function sets a value in the currently prompted field:

Syntax `fgl_dialog_setbuffer(var)` or `dialog.setbuffer(var)`

var Value or variable containing the value to be set in the current field

Returns None

Displaying a Row at a Given Line in a Screen Array

The following function displays a row of the program array to be set at a given line of the screen array:

Syntax `fgl_dialog_setcurrline (scr1, progl) or`
 `dialog.setcurrline (scr1, progl)`

scr1 Line of the screen array becoming current

progl Line of the program array becoming current

Returns None

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 the 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 fgl_dialog_setcurrline(5,100)
--#ON KEY(f5)
--#CALL fgl_dialog_setcurrline(1, 400)
END DISPLAY
CLOSE WINDOW w1
END MAIN
```


Compile this program with the following form:

```
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      ]
}
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
```

Displaying a Row at a Given Line in a Screen Array

After executing the program, an array appears with four buttons on the right side of the dialog box, as **Figure 5-6** shows. **Accept** and **Interrupt** are created automatically by the program due to the **fglprofile** configuration.

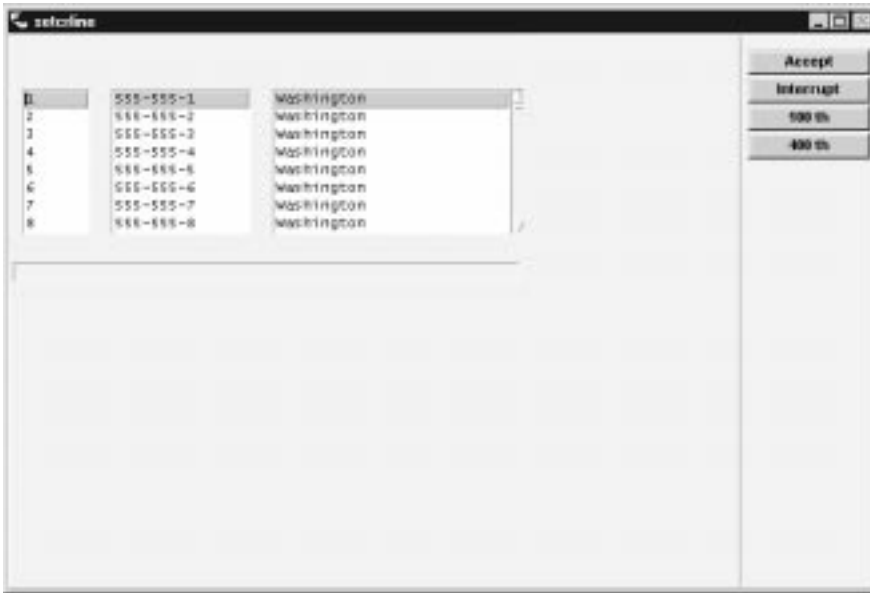


Figure 5-6
setcrline Dialog Box

If you press F4 or click the **100th** button, the screen array that [Figure 5-7](#) shows is displayed with the fifth row being current in the screen record displaying the 100th row from the program array.

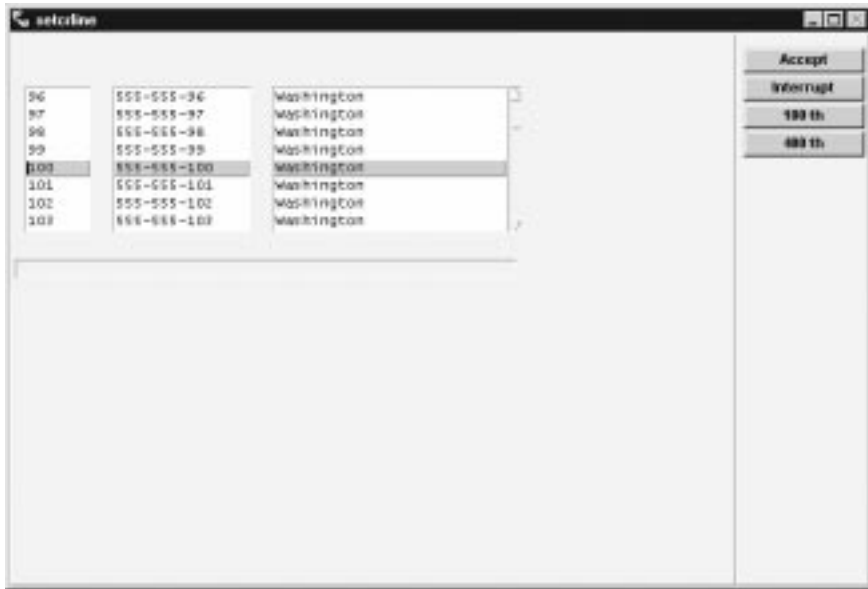


Figure 5-7
setcrlne Dialog Box

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.

Returning the Position of the Cursor

The following function returns the position of the cursor in the currently prompted field:

Syntax `fgl_dialog_getcursor()` or `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 **getcurs** button. The position of the cursor will be displayed in the error message list.

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_dialog_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
```

Setting the Cursor Position

The following function sets the cursor at a defined position in the currently prompted field:

Syntax `fgl_dialog_setcursor (pos) or 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.

Terminating Applications

The following option executes a 4GL function when the application window is closed by a user action, for example, ALT-F4 on Windows clients:

Syntax OPTIONS ON CLOSE APPLICATION {STOP|CONTINUE|
 CALL func}

Returns None

If a user tries to stop an application in graphical mode, the application either stops, continues, or a function is called.

If this option is not used by the program, the application displays a warning message that the application cannot be stopped:

Syntax OPTIONS ON CLOSE APPLICATION {WARN|SHOW
 WARNING}

Example OPTIONS ON CLOSE APPLICATION WARN

This option defines the function that must be called when the application receives the SIGTERM signal (only available on UNIX).

Syntax OPTIONS ON TERMINATE SIGNAL CALL func

Returns None

If this statement is not called, the program is stopped with the exit value of SIGTERM (15).



***Tip:** You can stop the program in a clean manner using ROLLBACK WORK. However, this does not have any user interaction.*

New Language Features

The following language features of INFORMIX-4GL were introduced in Version 7.30 (or in a few cases, in Version 7.20). All these features are supported by Dynamic 4GL, Version 3.0.

4GL 7.30 introduces new language features in several areas:

- Enhanced SQL syntax support
- Syntax for expansion of abbreviated year values
- Enhanced syntax for screen array management
- Dynamic configuration of report output
- New built-in operators
- New syntax to hide the comment line
- Editing multibyte data in 4GL forms
- New conditional comments
- Deprecated features

Enhanced SQL Syntax Support

Like all earlier releases since 4GL 4.10, this release supports most of the statement set of Version 4.1 of the Informix dialect of SQL language. These SQL statements can be directly embedded within 4GL source files. Statements and syntax enhancements added later than Version 4.10 of SQL must be prepared, if they are preparable.

Support For Embedded SQL 7.3 Syntax

4GL 7.30 supports all the directly embedded SQL statements that earlier releases of 4GL could embed, but also adds direct support for the following additional SQL statements:

- CONNECT
- CREATE PROCEDURE FROM
- DISCONNECT
- EXECUTE IMMEDIATE
- SET CONNECTION

4GL 7.30 also supports the following additional SQL syntax features:

- EXECUTE

The EXECUTE statement now supports the INTO and USING clauses in both orders:

```
{ { [INTO varlist1] [USING varlist2] } |
  { [USING varlist1] [INTO varlist2] } }
```

- FOREACH

The FOREACH statement now supports the WITH REOPTIMIZATION clause:

```
{ [USING varlist1] [INTO varlist2] WITH REOPTIMIZATION }
```

- OPEN

The OPEN statement now supports the WITH REOPTIMIZATION clause:

```
{ [USING varlist] WITH REOPTIMIZATION }
```

The non-keyword terms in these SQL statements can be specified as quoted strings or as character variables.

These SQL statements require an Informix database that recognizes them. When Dynamic 4GL accesses an Informix database earlier than Version 7.x, for example, the WITH REOPTIMIZATION clause has no effect.

Support for Preparable SQL Statements

Earlier releases of 4GL supported post-4.10 SQL syntax by the PREPARE feature, for SQL statements that can be prepared.

Continued Support Through PREPARE

4GL 7.30 continues to support the preparable SQL syntax of Informix 7.30 database servers. See the *Informix Guide to SQL: Syntax* description of PREPARE for a list of the SQL statements that cannot be prepared.

New SQL Statement Blocks

The same preparable statements that Section 2.1 describes are also supported in 4GL 7.30 by a new mechanism, SQL statement blocks, whose syntax resembles that of embedded SQL statements in ESQL/C:

```
SQL statement END SQL
```

Only a single preparable SQL statement can appear in each SQL block. If you delimit a preparable SQL statement by the keyword SQL before the SQL statement, and by the keywords END SQL after the SQL statement, then 4GL 7.30 prepares, executes, and frees the specified SQL statement when its SQL block is encountered, as in this example:

```
SQL
  BEGIN WORK
END SQL
```

Any 4GL variables that appear within an SQL block must be prefixed by the \$ symbol. One or more whitespace characters, such as blank spaces, can appear between \$ and the name of the host variable.

```
SQL
  UPDATE SomeTable
  SET (Co12, Co13, ... Co1N) = ($rec.co11 THRU $rec.co1N)
  WHERE CURRENT OF somecursor
END SQL
```

In the declaration of a database cursor, the following syntax that includes an SQL statement block within a DECLARE statement is valid:

```
DECLARE curs CURSOR
  SQL
  ... -- define the SELECT, UPDATE, or INSERT cursor
END SQL
```

An SQL block cannot appear, however, within a PREPARE statement.

SQL blocks of 4GL 7.30 (and Dynamic 4GL 3.0) support both singleton EXECUTE PROCEDURE statements that return values and singleton SELECT statements that return values, as in the following examples:

```
SQL
EXECUTE PROCEDURE someproc(1, $invar) INTO $var1, $var2
END SQL
```

```
SQL
SELECT Col1, Col2 INTO $var1, $var2
FROM SomeTable WHERE PKey = $var3
END SQL
```

The EXECUTE IMMEDIATE statement cannot appear within an SQL block.

Question-mark place-holders (?) in SQL blocks can appear in strings that are prepared, but not in other contexts. Thus, the following code generates a syntax error:

```
DECLARE cname CURSOR FOR
SQL
SELECT * FROM SomeWhere
WHERE SomeColumn BETWEEN ? AND ? -- Invalid!!!
END SQL
```

Trailing semicolon (;) symbols are valid after the SQL statement, but have no effect. Semicolons that separate two statements within the SQL block cause a syntax violation error message to be issued by the compiler. This causes the compilation to fail.

Optimizer directives and comments within delimited SQL statement blocks are passed to the database server, if you use the standard notation for these features in Version 7.30 and later Informix database servers. Such directives can immediately follow the DELETE, SELECT, or UPDATE keywords in SQL data manipulation statements. The plus (+) sign must be the first character following the comment indicator that begins an optimizer directive. The sharp (#) symbol is not a valid comment indicator in this context, but braces ({}) or double-hyphen (--) comment indicators are valid within an SQL block.

For more information, see the *Informix Guide to SQL: Syntax*.

Syntax for Expansion of Abbreviated Year Values

Some users (and some applications) abbreviate year values during data entry, so that, for example, September 9, 1999 might be entered as 9/9/99 (or with some other order of time units or time-unit separator symbols). Most earlier releases of 4GL expanded abbreviated year values by prefixing any 2-digit year with the two leading digits of the current year from the system clock.

Examples of statements of SQL and other 4GL statements and operators that can specify abbreviated year values include the following:

```
SELECT * FROM customer_name WHERE call_dtime
    BETWEEN (98-01 01) YEAR TO DAY AND DATETIME (04-12 31) YEAR TO DAY

UPDATE customer SET (city, reg_date)=("Palo Alto","12/01/04")
    WHERE customer_num = 103

INSERT INTO newtable SELECT dates FROM oldtable WHERE date1 "01/05/24"

DELETE FROM orders where order_date IN ("01/01/98", "12/31/04")

PREPARE up_sel FROM 'SELECT * FROM customer WHERE cust_date < "01/01/00"'

DEFINE cust_date DATE
    LET cust_date = "02/29/00"
    PROMPT for cust_date

DEFINE cust_date DATETIME YEAR TO DAY
    LET cust_date = "00-02-29"
    PROMPT for cust_time

DATE()
    DEFINE d DATE
    LET d = DATE ("02/29/00") ---- default DATE format
    LET d = DATE ("00-29-02") ---- format Y2MMDD-

EXTEND()
    DEFINE d1 DATETIME YEAR TO MINUTE
    LET d1 = EXTEND("02/28/00" YEAR TO MINUTE)

UNITS
    DEFINE d1 INTERVAL DAY TO DAY
    LET d1 = (DATE("03/01/00") - DATE("02/28/00")) UNITS DAY

WEEKDAY()
    DEFINE d1 INT
    LET d1 = WEEKDAY("02/28/00")

YEAR()
    DEFINE d1 INT
    LET d1 = YEAR ("02/28/00")
```

The **CONSTRUCT**, **INPUT**, and **INPUT ARRAY** statements can assign to variables the values that the user enters into the fields of the screen form.

Legacy Support for DBCENTURY

4GL 7.2 (and a few earlier releases to which this feature was back-ported) support the **DBCENTURY** environment variable, which can be set to any of four values, each of which specifies a different rule for expanding 2-digit years. (If the user enters a single-digit year, 4GL prefixes this digit with a leading zero, and then applies the current expansion rule to the 2-digit result.) This support is continued in 4GL 7.30.

You can set **DBCENTURY** as you would any other environment variable. The valid settings (and the expansion rule that each specifies) are listed here:

Setting	Expansion Algorithm
R	Prefix the entered value with the first two digits of the current year (from the system clock-calendar at runtime).
C	Prefix the entered value with the first two digits of the past, current, or future year that gives a date that is closest to the current date (from the system clock-calendar).
P	Prefix the entered value with the first two digits of the past (or current) year that gives the most recent date in the past.
F	Prefix the entered value with the first two digits of a future (or current) year that gives the earliest date in the future.

For example, on UNIX systems that use the C shell, the specification

```
setenv DBCENTURY=C
```

instructs 4GL to expand 2-digit years to the closest date.

DBCENTURY is case-sensitive. If **DBCENTURY** is not set, or set to an invalid value (such as any lowercase letter), then the default is **R**, which emulates the legacy behavior of most previous 4GL releases.

Unless you override **DBCENTURY** (as described in the next section), 4GL applies the **DBCENTURY** value that was in effect when program execution began to all year values (in **DATE** or **DATETIME** fields only) that have fewer than 3 digits; as with other environment variables, changing the **DBCENTURY** setting after program execution commences has no effect on any currently executing 4GL program. The results of using **DBCENTURY** are sensitive to the time of program execution and to the accuracy of the system clock-calendar.

DBCENTURY has no effect on fields that are not of the **DATE** or **DATETIME** data types, nor on **DATETIME** values that do not include **YEAR** as its first time unit, nor on year values that are not abbreviated. It can also affect data-type conversion of properly-formatted character strings that you assign to **DATE** or **DATETIME** variables.

To avoid expansion of years in the remote past that have only one or two digits, you must pad such values on the left with leading zeros, so that they have at least 3 digits.

New CENTURY Field Attribute

The **DBCENTURY** environment variable provides a global default rule for expanding 2-digit years. 4GL 7.3 supports a new field attribute in form specifications, called **CENTURY**, that provides the same functionality as **DBCENTURY** but at the field level of granularity. Unlike **DBCENTURY**, which provides a global expansion algorithm, different **DATE** or **DATETIME** fields can have different **CENTURY** settings. It supports the same **R**, **C**, **P**, and **F** settings as **DBCENTURY**, with the same semantics. It has this syntax:

```
CENTURY = { "F" | "C" | "P" | "R" }
```

Unlike **DBCENTURY**, the **CENTURY** field attribute is not case-sensitive. For example,

```
field-tag = ship_date, CENTURY = "C"
```

and

```
field-tag = ship_date, CENTURY = "c"
```

are equivalent. However, quotes are required around the setting.

If a DATE or DATETIME field has no CENTURY attribute, the **DBCENTURY** setting (or the **R** default, if **DBCENTURY** is not set) determines the expansion rule.

If CENTURY and **DBCENTURY** have different settings, then CENTURY takes precedence in the DATE and DATETIME fields that have this attribute.

Just as with **DBCENTURY**, the results of using CENTURY are sensitive to the time of program execution and the accuracy of the system clock-calendar.

CENTURY has no effect on fields that are not DATE or DATETIME data types, nor on DATETIME values that do not include YEAR as the first time unit, nor on unabbreviated year values. CENTURY can also affect conversion of properly-formatted character strings to DATE or DATETIME variables.

CENTURY is not needed for fields that display data from the database (as distinct from fields in which users enter data), because DATE columns of the database (and DATETIME columns that include the YEAR time unit) store only 4-digit years, even if a display field of the 4GL application is designed to show only trailing digits of YEAR values from the database.

New CENTURY Display Attribute in PROMPT Statements

The PROMPT statement of 4GL can request that the user enter a value. If the response value will be stored in a DATE or DATETIME variable, you can include CENTURY in the ATTRIBUTE clause that follows the FOR keyword, using the same semantics as in a form specification. The following PROMPT statement sets the expansion rule to the nearest date in the future:

```
DEFINE pasture DATE
PROMPT "When will you retire?" ATTRIBUTE (BLUE, REVERSE) FOR pasture
      ATTRIBUTE (GREEN, CENTURY = "F") ON KEY (F1) EXIT PROGRAM
END PROMPT
```

The value that follows the = symbol must appear within quotation marks, like the setting of the CENTURY field attribute.

Important: Dynamic 4GL requires Client SDK 2.30 to support the CENTURY attribute.



Enhanced Syntax for Screen Array Management

4GL 7.30 supports several new features that enhance the syntax of the INPUT ARRAY statement (and in some cases, DISPLAY ARRAY) to support program control over screen arrays.

Data Editing in Screen Arrays

New syntax has been added by which the programmer can prevent the user from performing Insert or Delete operations during the INPUT ARRAY statement. Such restrictions can be global to the entire screen array or can apply only to specific screen records.

New CANCEL INSERT Keywords

Insert operations of the user can be cancelled programmatically for individual screen records of the current 4GL form by including the CANCEL INSERT keywords within the BEFORE INSERT control block. The cancelled Insert operation has no effect on the active set of rows that INPUT ARRAY is processing.

The new syntax for the BEFORE INSERT control block of INPUT ARRAY statements is:

```
INPUT ARRAY ...  
  BEFORE INSERT statement... { CANCEL INSERT }
```

Here *statement* is any statement of 4GL that is valid within a BEFORE INSERT control block of INPUT ARRAY.

If CANCEL INSERT is specified, the user is prevented from entering rows by using the Insert key. This feature also prevents the user from entering rows by moving the screen cursor past the last initialized row by using an ARROW key, TAB key, RETURN key, or (in Dynamic 4GL) the ENTER key.

Here is a code example:

```
INPUT ARRAY ...  
BEFORE INSERT  
  IF ARR_CURR() == 3  
  THEN  
    CANCEL INSERT  
  END IF  
END INPUT
```

This example disables the Insert key for only the third row.

In contexts like this, two cases arise:

- **Case 1: The form is already populated with data.**
Suppose that it contains five rows filled with data. If the cursor comes to the third (populated) row and F1 is pressed, a new row is not inserted, because CANCEL INSERT prevents any Insert operation for this row.
- **Case 2: Only some of the rows are filled with data.**
Suppose that only two of a possible five rows contain data. In this case, the user cannot move to the third row using the ARROW, TAB or RETURN key, because CANCEL INSERT prevents any Insert operation for this row.

For more information about CANCEL INSERT, see [“New CANCEL DELETE Keywords” on page 5-39](#).

New CANCEL DELETE Keywords

Delete operations by the user can also be cancelled programmatically for individual screen records of the current 4GL form by including the CANCEL DELETE keywords within the BEFORE DELETE control block. The cancelled Delete operation has no effect on the active set of rows that INPUT ARRAY is processing.

The new syntax for the BEFORE DELETE control block of INPUT ARRAY statements is:

```
INPUT ARRAY ...
    BEFORE DELETE statement... { CANCEL DELETE }
```

Here *statement* is any statement of 4GL that is valid within a BEFORE DELETE control block of INPUT ARRAY.

If CANCEL INSERT or CANCEL DELETE is executed, the current BEFORE INSERT or BEFORE DELETE control block is terminated, and control of program execution passes to the next statement that follows the terminated control block.

As an example, the programmer might want to implement a system where the user is allowed to delete all but one of the rows, but once a row is deleted, a replacement row cannot be inserted in its place. The following code implements this design:

```
DEFINE n_rows INTEGER
DEFINE arrayname ARRAY[100] OF RECORD
. . .

INPUT ARRAY arrayname WITHOUT DEFAULTS FROM s_array.*
  ATTRIBUTES(COUNT = n_rows, MAXCOUNT = n_rows,
    INSERT ROW = FALSE, DELETE ROW = TRUE)

BEFORE INSERT
  CANCEL INSERT

BEFORE DELETE
  LET n_rows = n_rows - 1
  IF n_rows <= 0 THEN
    CANCEL DELETE
  END IF

END INPUT
```

New INSERT ROW Attribute

4GL 7.30 supports another new syntax feature that provides a means by which the programmer can enable or disable Insert operations for the entire form during INPUT ARRAY statements. The new INSERT ROW attribute can be set to TRUE or FALSE in the ATTRIBUTE clause that follows the INPUT ARRAY binding clause.

The new attribute has this syntax:

```
INSERT ROW [ = { TRUE | FALSE } ]
```

When INSERT ROW = FALSE is specified, the user cannot perform any Insert actions within the INPUT ARRAY statement.

For additional information about the INSERT ROW attribute in INPUT ARRAY statements, see [“New DELETE ROW Attribute” on page 5-41](#).

New DELETE ROW Attribute

4GL 7.30 also supports a new syntax feature that provides a means by which the programmer can enable or disable Delete operations for the entire form during INPUT ARRAY statements. The new DELETE ROW attribute can be set to TRUE or FALSE in the ATTRIBUTE clause that follows the INPUT ARRAY binding clause.

The new attribute has this syntax:

```
DELETE ROW [ = { TRUE | FALSE } ]
```

When DELETE ROW = FALSE is specified, the user cannot perform any DELETE actions within the INPUT ARRAY statement.

When INSERT ROW = TRUE or DELETE ROW = TRUE is specified, then the user is not prevented from performing the action for which TRUE is specified.

The default in both cases is TRUE, which corresponds to the legacy behavior of previous 4GL releases.

In Dynamic 4GL 3.0, these attributes have an extended syntax:

```
INSERT ROW [ = { TRUE | FALSE | var } ]
DELETE ROW [ = { TRUE | FALSE | var } ]
```

Here *var* is a variable that contain a Boolean value. If the value of *var* is zero or FALSE or NULL, then the value of the attribute is FALSE. For other values of *var* (or by default, if no value is specified) the value of the attribute is TRUE.

The following example disables Insert and Delete operations on rows of the screen array:

```
INPUT ARRAY arrayname WITHOUT DEFAULTS FROM s_array.*
  ATTRIBUTE(INSERT ROW = FALSE, DELETE ROW = FALSE)
```

New CURRENT ROW DISPLAY Attribute

4GL 7.30 has added a new attribute to the DISPLAY ARRAY and INPUT ARRAY statements. This attribute enables the 4GL programmer to highlight the current row of a screen array. Dynamic 4GL always highlights the current row in GUI mode, so this new attribute is primarily useful in character-based deployment.

For both DISPLAY ARRAY and INPUT ARRAY, the ATTRIBUTE clause can now include the following syntax:

```
CURRENT ROW DISPLAY = "string"
```

Here *string* is a comma-separated list of screen attributes. The *string* can include zero or more intensity attributes from among the following:

- REVERSE
- UNDERLINE
- BOLD
- BLINK

The *string* can also include one or none of the color attributes from among the following:

- BLACK
- BLUE
- CYAN
- GREEN
- MAGENTA
- RED
- WHITE
- YELLOW

These attributes are applied to the current row of the screen array. The contents of *string* is not case-sensitive.

An error is issued if *string* is an empty string:

```
CURRENT ROW DISPLAY = ""      --Error!
```

The following statement sets the CURRENT ROW DISPLAY attribute:

```
INPUT ARRAY arrayname WITHOUT DEFAULTS FROM s_array.*  
  ATTRIBUTE (BOLD, CURRENT ROW DISPLAY = "RED, REVERSE", UNDERLINE)
```

The rows other than the current row in this array are displayed in bold and underlined, but the current row is displayed in red and in reverse video. If there is only one row in the screen array, then the current row is the only one that is displayed.

New COUNT Attribute

4GL 7.30 supports two additional new attributes in the ATTRIBUTE clause of INPUT ARRAY statements, in order to provide dynamic control of input to screen arrays.

The COUNT attribute can specify the number of records within a program array that contain data. It can appear within the ATTRIBUTE clause of the INPUT ARRAY statement.

In 4GL 7.30 and Dynamic 4GL 3.0, COUNT has the following syntax:

```
COUNT = { n | var }
```

Here *n* is a literal integer, and *var* is an INTEGER or SMALLINT variable. The specification:

```
COUNT = 5
```

is equivalent to the 4GL statement:

```
CALL SET_COUNT(5)
```

Both of these specifications restrict the number of screen records that can be displayed in the current screen array to 5.

New MAXCOUNT Attribute

The MAXCOUNT attribute can specify the dynamic size of a screen array. This can be less than the declared size that the INSTRUCTIONS section of the .per file specifies for the screen array. MAXCOUNT is valid only within the ATTRIBUTE clause of the INPUT ARRAY statement.

In 4GL 7.30 and in Dynamic 4GL 3.0, MAXCOUNT has this syntax:

```
MAXCOUNT = { n | var }
```

Here *n* is a literal integer, and *var* is an INTEGER or SMALLINT variable. The following example shows an INPUT ARRAY statement that specifies both the MAXCOUNT and COUNT attribute:

```
INPUT ARRAY prog_array WITHOUT DEFAULTS
FROM scr_array.* ATTRIBUTE( MAXCOUNT = x, COUNT = y )
```

Here *x* and *y* are literal integers or integer variables. In this example, *y* is the number of records that contain data within the program array. The MAXCOUNT value of *x* determines the dynamic size of the screen array that displays the program array.

If MAXCOUNT is specified as less than one or greater than the declared program array size, then the original program array size is used as the MAXCOUNT value.

Both COUNT and MAXCOUNT can be specified in the same ATTRIBUTE clause:

```
CALL SET_COUNT(5)
INPUT ARRAY prog_array WITHOUT DEFAULTS
FROM scr_array.* ATTRIBUTE(MAXCOUNT = 10, COUNT = 6)
```

In this case, the COUNT attribute overrides the SET_COUNT value. The number of rows displayed will be 6.

New FGL_SCR_SIZE() Built-In Function

4GL 7.30 supports a new built-in function that accepts as its argument the name of a screen array, and returns an integer that corresponds to the number of screen records in that screen array.

It has this calling syntax:

```
FGL_SCR_SIZE ( { "array" | var } )
```

Here *array* is the name of the screen array, as declared in the INSTRUCTIONS section of the form specification. This can appear in the function call as an identifier enclosed between quotation marks, or as a character variable containing the name of the screen array.

The following form-specification file (called **file.per**) declares a screen array that the subsequent 4GL code example references:

```
DATABASE FORMONLY

SCREEN
{
[f1   ] [f2   ]
[f1   ] [f2   ]
[f1   ] [f2   ]
[f3   ] [f4   ]
[f3   ] [f4   ]
[f5   ]
}

ATTRIBUTES
f1 = FORMONLY.a ;
f2 = FORMONLY.b ;
f3 = FORMONLY.c ;
f4 = FORMONLY.d ;
f5 = FORMONLY.e ;

INSTRUCTIONS
DELIMITERS ""
SCREEN RECORD s_rec1[3] (a,b)
SCREEN RECORD s_rec2 (c,d)
```

The following 4GL program invokes the **FGL_SCR_SIZE()** function:

```
MAIN

DEFINE n1,n2 INT
DEFINE ch CHAR(10)

OPEN WINDOW w1 AT 2,3 WITH FORM "file" ATTRIBUTE (BORDER)
CALL fgl_scr_size("s_rec1") RETURNING n1
LET n1 = fgl_scr_size("s_rec1") -- Can also be called
                                -- in a LET statement
DISPLAY "n1 = ", n1

LET ch = "s_rec2"
CALL fgl_scr_size(ch) RETURNING n2
LET n2 = fgl_scr_size(ch) -- Can also be called
                                -- in a LET statement
DISPLAY "n2 = ", n2
CLOSE WINDOW w1
END MAIN
```

This example produces the following output:

```
n1 = 3  
n2 = 2
```

The proper value is returned, even though the array dimension is not specified.

An error is returned if no form is open, or if the specified screen array is not in the current open form.

Dynamic Configuration of Report Output

All earlier releases of 4GL required that the parameters of the OUTPUT section of a report be specified as fixed values (either literal integers or string literals) in the REPORT definition.

A new feature of this release enables you to redefine all the values of the OUTPUT section of a report in the START REPORT statement, so that the destination and dimensions of output from the report can be specified at runtime.

If you do not use START REPORT to specify these parameters, then the OUTPUT section values (or default values, if the OUTPUT section omits any specification) remain in effect, as in previous releases.

The syntax of `START REPORT` in 4GL 7.30 and Dynamic 4GL 3.0 is:

```
START REPORT report
[TO {
  SCREEN                |
  PRINTER               |
  FILE {"file"|var}    |
  {"file"|var}         |
  PIPE [ IN {FORM|LINE} MODE] {"program"|var} |
  OUTPUT {"out"|var} [DESTINATION {"trg"|var}]
}
]
[WITH {
  TOP OF PAGE    = "string" |
  PAGE LENGTH   = n |
  TOP MARGIN     = n |
  BOTTOM MARGIN  = n |
  LEFT MARGIN    = n |
  RIGHT MARGIN   = n |
  [...]]
}
```

Here *n* is an integer expression, and each instance of *var* is a 4GL program variable whose logical content is the string at the left of the preceding pipe (|) symbol.

Here *out* must be one of the following:

```
"SCREEN", "PRINTER", "FILE", "PIPE [ IN {FORM|LINE} MODE]"
```

These *out* keywords are not case-sensitive.

Here *trg* is the name of a file or program to receive the report output from a pipe. If *out* has the value `SCREEN` or `PRINTER`, then the `DESTINATION` clause is not needed (and is ignored, if specified).

The `WITH` clause supports the `TOP OF PAGE = "string"` option, which substitutes a page-eject string for repeated Linefeeds to complete the current page of report output and begin the next page. See your printer documentation for the appropriate string value. Only the first character in the string is passed to the printer. (This feature can reduce the time required to print long reports.)

Example:

```
START REPORT repname TO OUTPUT string1 DESTINATION string2
  WITH TOP MARGIN = 2, TOP OF PAGE = "^L", PAGE LENGTH = 66
```

Here the caret (^) symbol specifies the CONTROL key; this is in contrast with other 4GL features (for example, the ON KEY clause) that use "CONTROL - " as the notation to reference the CONTROL key.

If the WITH clause appears, its values override any corresponding parameter that were explicitly stated in the OUTPUT section of the REPORT definition, or any default values.

An implication of this new feature is that any report can now use either top-of-page character string processing or repeated blank lines to space to the page, depending on what was specified when the report is started.

New Built-In Operators

This release supports a new string concatenation operator, resembling that of the Informix dialect of SQL. It also supports a new synonym for the "=" relational operator, resembling the "==" operator of C.

String Concatenation Operator

In all releases of 4GL, the comma (,) symbol has concatenation semantics in some contexts for lists of strings (as in LET, PREPARE, and PRINT statements). 4GL 7.30 introduces a double-pipe (||) concatenation operator that accepts two values of simple data types as operands, and joins them to return a single character-string value.

In the right-hand side of the LET statement, or in the argument list of a function call, the concatenation operator can join two values of any simple data type. It associates its operands from left to right. Thus, (a || b || c) and ((a || b) || c) are equivalent. Precedence of this operator is higher than LIKE, MATCHES, or relational operators, but lower than the arithmetic operators.

If either operand is a NULL string, then the returned value is NULL, represented as a string of zero length. This is in contrast to how LET ignores NULL values within comma-separated lists, unless every value in the list is NULL. (The NULL and LET returns from a comma-separated list of NULL values is represented as a single blank space.)

For example, if *a* and *b* are non-NULL strings, and *c* is NULL, then

```
LET x = a,b,c-- This assigns (a || b) to variable x.  
LET y = a || b || c-- This assigns NULL to variable y.
```


In LET statements, you can choose between these two methods of treating NULL values in concatenation by substituting || for comma if you want a NULL string returned from any concatenation that includes a NULL operand.

In some contexts, only || can perform concatenation.

```
CALL myfunct ( a || b, c )
```

is not equivalent to

```
CALL myfunct ( a, b, c )
```

because comma is always a list separator symbol in function calls.

Concatenation with the || operator discards trailing whitespace from operands of integer and fixed-point number data types, but not from character or floating point data types. The CLIPPED operator of 4GL can remove trailing blanks from values before concatenation in 4GL statements, but TRIM must replace CLIPPED in preparable SQL statements (for Version 7.x and later Informix databases).

Synonym for the Equality (=) Relational Operator

The CONSTRUCT statement and Boolean expressions of 4GL support relational operators that can perform comparisons of two data values.

4GL 7.3 now supports "==" as a synonym for "=" (just as earlier releases have supported "!=" as a synonym for the "<>" operator).

For example, the expressions in the two statement fragments:

```
IF x = y THEN GO TO :finario
IF x == y THEN GO TO :finario
```

are equivalent.

New Syntax to Hide the Comment Line

By default, the last line of the current 4GL window is the Comment line, which can display messages that the COMMENT attribute of a 4GL form specifies. This is a reserved line, which is cleared when the user moves the visual cursor to a new line of a screen form, so it is typically used to send messages to the user, rather than for data entry or data display. In 4GL forms that do not use the COMMENT attribute, the Comment line is unused space on the screen.

4GL 7.30 introduces a new feature that enables you to conserve display space within a 4GL window by hiding the Comment line. The new syntax that can appear within the ATTRIBUTE clause of the OPEN WINDOW statement is:

```
COMMENT LINE OFF
```

This is supported by both 4GL 7.30 and by Dynamic 4GL 3.0. If this is specified, then the Comment line is hidden for that 4GL window and cannot display messages from the form specification, even if some fields of a form that this window displays have the COMMENT attribute.

Dynamic 4GL can use a new syntax extension

```
COMMENT LINE = { n | var }
```

where *n* is a digit in the range from 0 to the number of lines in the 4GL window, and *var* is an integer variable with the same range. The Dynamic 4GL specification

```
COMMENT LINE = 0
```

in the ATTRIBUTE clause of the OPEN WINDOW statement is logically equivalent to the new 4GL 7.30 specification:

```
COMMENT LINE OFF
```

Editing Multibyte Data in 4GL Forms

4GL 7.20 introduced GLS, a locale-based feature for supporting the entry, display, retrieval, and collation of strings that include non-ASCII characters, as well as display formats for number, time, and currency data for various languages and cultures besides those of the default (U.S. English) locale. 4GL 7.x supports the Informix locale files for most European and Asian languages, including multibyte East Asian locales for Chinese, Japanese, and Korean languages.

4GL does not, however, support languages that use right-to-left or bidirectional text (such as Arabic, Farsi, Hebrew, or Urdu).

Additional details about GLS can be found in the *Informix Guide to GLS Functionality*.

The following environment variables can be set to support non-default character sets, and cultural conventions for the display of numeric, date, and currency data values:

- **CLIENT_LOCALE**
- **DBAPICODE**
- **DB_LOCALE**
- **DBNLS**
- **GL_DATE**
- **GL_DATETIME**
- **LANG**
- **SERVER_LOCALE**

SQL identifiers that include non-ASCII characters that the locale of the database supports can appear within SQL statements in 4GL source code, provided that the locale of the 4GL client system also supports these non-ASCII characters in its codeset.

4GL identifiers, data strings, and the values of CHAR, VARCHAR, and TEXT variables, formal arguments, and returned values can include non-ASCII characters that the current locale supports.

4GL forms can include locale-dependent characters as text, and fields can support such characters in data entry and data display operations. For East-Asian locales that support multibyte characters, this feature is new in 4GL 7.30. If a multibyte string requires more bytes of storage than the declared size of a 4GL form field, or more than a segment of a field, then the string is truncated from the right. Any partial character that might be created by this truncation is replaced by whitespace.

In multibyte locales, the storage length (in bytes) can be longer than the display length of the field in a form. For example, a form field that is declared LIKE a CHAR(16) database column cannot display a string that consists of 16 multibyte characters because such a string occupies at least 32 bytes of storage.

In locales whose codesets include multibyte characters, 4GL does not create partial characters. In 4GL or SQL operations that attempt to divide a string within a multibyte logical character, whitespace is substituted for any partial character. Examples of operations in which whitespace automatically replaces any partial characters include 4GL expressions that include the substring ([]) operator, truncation of data strings when they are stored in variables, and data entry into fields of 4GL forms in which the length (in bytes) of the field is smaller than the length (in bytes) of the data string.

The following built-in functions and operators of 4GL can accept or return locale-supported non-ASCII (and multibyte) characters, or can process multibyte characters without creating partial characters:

- CLIPPED operator
- DOWNSHIFT()
- LENGTH()
- Substring ([]) operator
- UPSHIFT()
- WORDWRAP operator

In this release, however, the built-in functions **FGL_GETENV()** and **FGL_KEYVAL()** cannot return multibyte characters. (They can return single-byte non-ASCII characters that the client locale supports.)

In Japanese locales, WORDWRAP fields in 4GL forms and in output from reports perform single-pass kinsoku processing, for characters that the locale file lists as prohibited from appearing at the beginning or end of a line. If a character that is prohibited from ending a line appears at the end of a line, it is moved down to the beginning of the next line. A character that precedes another that is prohibited from beginning a line can similarly be moved down to the next line. (By *single pass* is meant that each line is tested only once. Even if this process results in a line ending in a forbidden character, no further kinsoku processing is performed.) The locale files must identify the characters that are prohibited from beginning a line or prohibited from ending a line.



Important: *Dynamic 4GL requires Client SDK 2.30 to support these GLS features.*

New Conditional Comments

In Dynamic 4GL, you can write:

```
--# CALL fgl_init4js()
```

The Dynamic 4GL compiler treats the '--#' as whitespace and ignores it, compiling a call to the **fgl_init4js()** function. 4GL sees the '--' as a comment indicator and treats the rest of the line as a comment.

4GL 7.30 introduces an analog to this Dynamic 4GL feature, providing a notation that 4GL treats as whitespace and ignores, compiling what follows as ordinary code, but Dynamic 4GL sees the line as a comment.

This is the notation:

```
--@
```

This allows the programmer to write:

```
--# OPTIONS HELP FILE "d4gl_help.42h" --Line is ignored by I-4GL
--@ OPTIONS HELP FILE "i4gl_help.iem" --Line is ignored by D-4GL
```

The previous two lines are the logical equivalent to the following logic:

```
--# IF TRUE THEN
--# OPTIONS HELP FILE "d4gl_help.42h"
--# ELSE
    OPTIONS HELP FILE "i4gl_help.iem"
--# END IF
```

This is much more verbose and makes less clear what is intended.

Conditional comments are supported both in source (.4gl) files and in form specification (.per) files. The following example shows a fragment of a form specification that uses one attribute for the 4GL form compiler, and another for the Dynamic 4GL form compiler:

```
ATTRIBUTES
  f0 = FORMONLY.name, --#char_var ;
  --@REVERSE ;
```

They are also valid within SQL statement blocks but not within the text of a PREPARE statement.

These symbols are called *conditional comment indicators* because their effect depends on whether you compile with the 4GL or Dynamic 4GL compiler. Just as with other comment indicators, they have no special significance within a quoted string.

Because the compilers treat a conditional comment as either whitespace or as a comment, you cannot use both in the same line, as in this example:

```
CALL abc--#function()--@procedure()
```

Conditional comment indicators are treated as whitespace if the compiler does not treat them as beginning a comment, so the previous example always generates a compile-time syntax error, because neither 'CALL abc' (using 4GL) nor 'CALL abc procedure()' (using Dynamic 4GL) is valid code. When a conditional comment indicator is interpreted as beginning a comment, the comment marker '--#' or '--@' terminates the previous keyword or symbol.

Because "--@" is a new syntax feature of 4GL 7.30, it is treated as a comment symbol by all previous releases of the 4GL (and Dynamic 4GL) compilers, just as "--#" is treated as a comment symbol in all 4GL releases.

Using Form Extensions to 4GL

In This Chapter	6-3
Implementing List Boxes	6-4
Implementing Buttons.	6-6
Menu Buttons	6-6
Hot-Key Buttons	6-6
Editing fglprofile	6-7
Editing the .per File	6-7
Setting the KEY Field Attribute	6-8
Using 4GL Functions	6-8
Buttons in the Form	6-9
Implementing Bitmaps	6-11
Implementing Check Boxes and Radio Buttons	6-11
Check Box Syntax	6-11
Radio Button Syntax	6-12
Invoking a Key Code	6-13
Combo Fields.	6-14
Implementing Scrolling Fields	6-15
Creating Folder Tabs	6-16

In This Chapter

This chapter describes the 4GL language extensions that Dynamic 4GL added and that you can add to the form-specification files. The following sections describe the 4GL extensions for implementing these controls:

- List boxes
- Command buttons
- Bitmaps
- Check boxes
- Radio buttons
- Combo boxes
- Scrolling fields
- Folder tabs

Implementing List Boxes

Screen arrays in graphical mode are displayed with list box objects. The following code generates output in graphical mode, as [Figure 6-1](#) shows:

```
DATABASE formonly
SCREEN {

    [f01                ] [f02                ]
    [f01                ] [f02                ]
    [f01                ] [f02                ]
    [f01                ] [f02                ]
    [f01                ] [f02                ]
    [f01                ] [f02                ]
    [f01                ] [f02                ]
    [f01                ] [f02                ]
    [f01                ] [f02                ]
    [f01                ] [f02                ]
    [f01                ] [f02                ]

}
ATTRIBUTES
f01 = formonly.f01 type char;
f02 = formonly.f02 type char;
INSTRUCTIONS
SCREEN RECORD s_rec[10] (f01,f02):
```

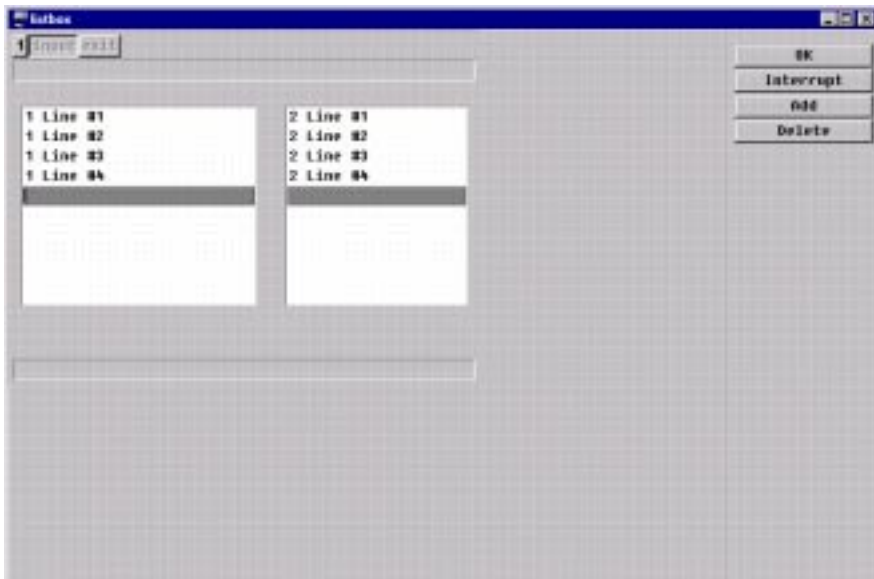


Figure 6-1
Graphical Mode
Output

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 results that [Figure 6-2](#) shows.

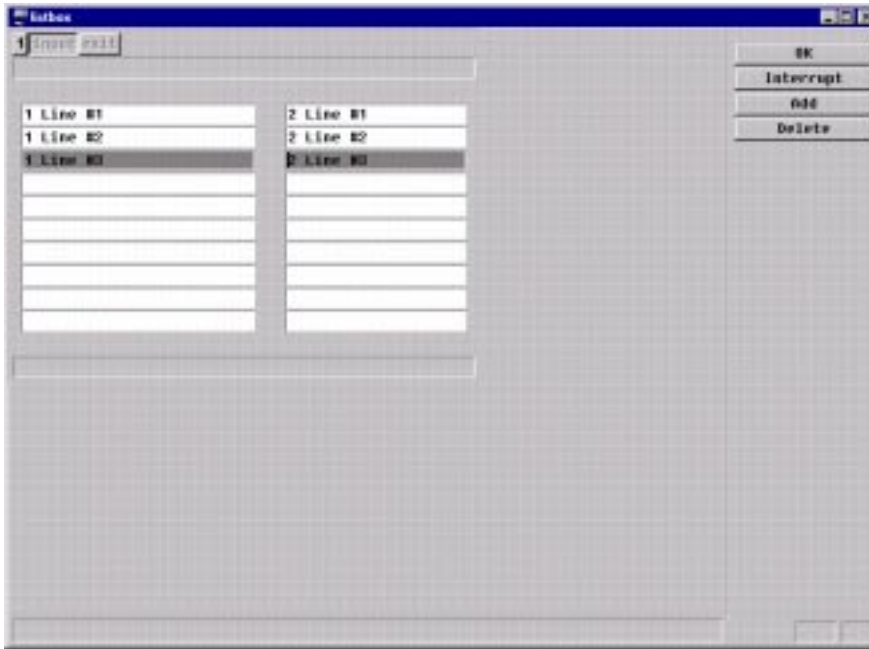


Figure 6-2
Graphical Mode
Output with Fields
on Separate Lines

You can also make fields appear on individual lines by including the following setting in your **fglprofile** file:

```
Resource.gui.workScreenPlace.nolist=1
```

This feature might 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.

Implementing 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.

To choose the positioning of the button on the screen, use the **Menu.style** resource in the **fglprofile** configuration file, as follows:

`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, “Using Non-Graphical Extensions to 4GL,”](#) and [Appendix B, “Common Problems and Workarounds.”](#)

Hot-Key Buttons

Hot keys that you define in `COMMAND KEY` or `ON KEY` statements are displayed in a separate frame located on the right side of the application. These buttons automatically appear when activated.

To access hot keys, press the corresponding key with the ASCII version of the application or click the button with the mouse.

To edit the labels of hot-key buttons, use the following order of precedence, listed from highest to lowest:

1. The KEY attributes in a field of a **.per** file. (You cannot change a label with **fgl_dialog_setkeylabel()** if the same key attributes are present for a special key.)
2. The **fgl_dialog_setkeylabel()** function
3. The KEYS section in a **.per** file
4. The **fgl_setkeylabel** function
5. The **fglprofile** file

Editing fglprofile

The **\$FGLDIR/etc/fglprofile** configuration 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"
```

The order of appearance of the hot key in the right frame is defined by the `key."key_name".order` resource in **fglprofile**.

Editing the .per File

This method edits the KEYS section in the form-specification file (**.per** file), to display the label of a hot-key button when the corresponding form is used. For 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.

Setting the KEY Field Attribute

This 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. Use the following syntax:

```
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**.

Using 4GL Functions

This method uses calls to 4GL functions in your 4GL source-code modules. These functions are divided into the following two categories:

- Functions that execute during a dialog with the user; for example, during INPUT, INPUT ARRAY, or CONSTRUCT.
- Functions that are not specific to the current user dialog.

If you want to change a label for a specific user dialog, use the **fgl_dialog_setkeylabel()** function. Use the following syntax:

```
fgl_dialog_setkeylabel("key", "label")
```

If you want to change the label outside a specific user dialog, you must do so before the beginning of the dialog statement. Use the following syntax:

```
fgl_setkeylabel(hot_key_name, new_label)
```

or

```
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:

```

...
BEFORE INPUT
CALL FGL_SETKEYLABEL ("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**. 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 (undefined in this example).

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 **fglprofile**.

An empty button does not react to mouse clicks. This behavior is defined in the **fglprofile** configuration file with the following resource:

<code>gui.empty.button.visible = 1</code>	The button remains visible but does not react to mouse clicks. This is the default value.
<code>gui.empty.button.visible = 0</code>	The button becomes invisible and disappears.

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.

Buttons in the Form

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 **widget** and the **config** string in the attribute definition of the tag. The **widget** parameter must be set to **BUTTON** and the **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:

```
.
.
.
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.
```

Create this form-specification file, **button.per**:

```
DATABASE formonly
SCREEN {
  Field1 [a]
  Field2 [b]
  Field3 [c]
        [bt  ] [bt2  ]
}
ATTRIBUTES
a = formonly.a;
b = formonly.b;
c = formonly.c;
bt1 = formonly.bt1
--# , widget="BUTTON", config="Control-a"
;
bt2 = formonly.bt2
--# , widget="BUTTON", config="F1-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.

Implementing 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.

Implementing Check Boxes and Radio Buttons

You can use check boxes 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 are 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 are displayed.

In the file `check.per`:

```
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
```

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 parameters 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.



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 displays frames that include three radio buttons.

In the file `radio.per`:

```

DATABASE formonly
screen {

    radiobutton:
        [rad001          ]

}
attributes

rad001 = formonly.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 and **lab_two** for the second button.

The value returned by the radio button is a null string if no button is selected in the frame. It is possible to define a default value for the radio button group.

Invoking a Key Code

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 radio button or check box in the form file, as the following example shows:

```

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 {Radio #1} F12 {Radio #2}
F13 {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 be unable to use it.

Combo Fields

The **combo** field object is an association between a **classical** field and a bitmap (**bmp** field) 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:

```

DATABASE formonly
screen {
    bmp field:
    [bmf001      ]
}
attributes
bmf001 = formonly.bmp_field, widget="FIELD_BMP", config="combo.bmp
Control-q";
end

```

Implementing 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:

```

MAIN
DEFINE text CHAR(512)
OPEN WINDOW w1 AT 1,1 WITH FORM "demo1"
INPUT BY NAME text
CLOSE WINDOW w1
END MAIN

```

- In the **.per** file:

```

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.

Creating Folder Tabs

Folder tabs allows you to create tabs that display different parts of a form. For example, you might divide a form for entering information into three subforms that you can display by clicking a folder tab.

Important: *You do use an input statement on fields located on different subforms.*

To use this feature, add more than one SCREEN section in a form. To set the label in the folder tab, use the following syntax:

```
SCREEN TITLE "label"  
{  
  ...  
}
```

with the label appearing as the name of the folder tab.

The following example of how to create folder tabs shows two files, **demo1.per** and **demo1.4gl**. When you compile and run these files, the input and input array buttons display a form as three subforms. The input is done through three fields on the first two subforms and the input array is done on the third subform. The code is as follows:



File demo1.per:

```
DATABASE formonly
SCREEN TITLE "screen 1/3"
{
    field 1 [f01                ]
    field 2 [f02                ]
}
SCREEN TITLE "Screen 2/3"
{
    field 3 [f03                ]
}
SCREEN TITLE "Screen 3/3"
{
    Array row 1 [a01            ]
    Array row 2 [a01            ]
    Array row 3 [a01            ]
    Array row 4 [a01            ]
}
ATTRIBUTES
f01 = formonly.f01;
f02 = formonly.f02;
f03 = formonly.f03;
a01 = formonly.a01;

INSTRUCTIONS
screen record scr_arr[4] (a01)
```

File demo1.4gl:

```
MAIN

DEFINE f01,f02,f03 CHAR(20)

DEFINE arr ARRAY[10] OF RECORD
  a01 CHAR(10)
END RECORD

OPEN FORM frm1 from "demo1"

MENU "Folder tabs"
  COMMAND "Input"
    OPEN WINDOW w1 AT 3,1 WITH 25 rows, 80 columns
    DISPLAY FORM frm1
    INPUT BY NAME f01, f02, f03
    CLOSE FORM frm1
    CLOSE WINDOW w1
  COMMAND "Input array"
    OPEN WINDOW w1 AT 3,1 WITH 25 rows, 80 columns
    DISPLAY FORM frm1
    INPUT ARRAY arr from scr_arr.*
    CLOSE FORM frm1
    CLOSE WINDOW w1
  COMMAND "Exit"
    EXIT MENU
END MENU
END MAIN
```


Using Graphical Extensions to 4GL

In This Chapter	7-3
Display Extensions	7-3
Calling Dynamic 4GL Libraries	7-3
Checking for UNIX or Windows	7-4
Checking for Windows Client Mode	7-5
Window-Management Functions	7-6
Setting the Default Size of a Window	7-6
Setting the Title of a Window	7-7
Retrieving Information from a Field.	7-8
Retrieving Information from an Application Window	7-8
Setting the Active Window.	7-10
Closing a Window.	7-10
Creating Toolbars and Toolbar Icons	7-11
Creating Dialog Boxes.	7-12
Creating an Interactive Message Box	7-12
Displaying an Interactive Message Box	7-14
Formatting Text in a Message Box	7-15
Entering a Field Value into a Message Box	7-16
Using Drawing Extensions	7-17
Mouse-Management Functions	7-18
Returning a Value After a Left Mouse Click.	7-18
Returning a Value After a Right Mouse Click	7-18
Remove Key Binding	7-19
Defining the Drawing Area	7-19
Initializing the Drawing Function	7-20
Selecting a Drawing Area	7-20

Specifying the Text Insertion Point	7-21
Setting Line Width	7-22
Clearing the Draw Function	7-22
Drawing Rectangles	7-23
Setting the Fill Color	7-23
Drawing an Oval	7-23
Drawing a Circle	7-24
Drawing a Line	7-24
Drawing Text	7-25
Drawing an Arc.	7-25
Drawing a Polygon	7-26

In This Chapter

This chapter describes functions that can be used to enhance the graphical user interface (GUI). The topics include:

- Display extensions
- Window-management functions
- Mouse-management functions
- Toolbars
- Dialog boxes
- Drawing extensions

Display Extensions

This section describes an initialization function and other functions that relate to display environments.

Calling Dynamic 4GL Libraries

The following function is required at the beginning of every 4GL program that calls functions from the Dynamic 4GL libraries:

Syntax `fgl_init4js()`

For example:

```
--#CALL fgl_init4js()
```

Checking for UNIX or Windows

The following function returns the current value of the **FGLGUI** environment variable:

Syntax `fgl_fglgui()`

Returns **TRUE** if the program is run under a GUI
 FALSE if the program is run on an ASCII terminal

For example:

```
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.

[Figure 7-1](#) shows the message that appears if you are in ASCII mode for UNIX.

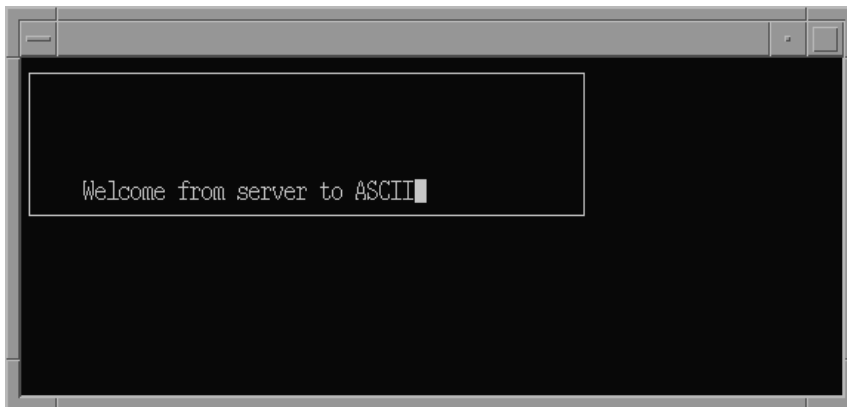


Figure 7-1
*Welcome Message
in ASCII Mode*



Tip: If you execute this program with UNIX, be sure to put a **SLEEP** statement after **DISPLAY** so that you can see the message.

Figure 7-2 shows the message that appears if you are in graphical mode.



Figure 7-2
Welcome Message
in Graphical Mode

Checking for Windows Client Mode

The following 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 GUI and, if so, whether you are using Windows:

```

MAIN
IF fgl_fglgui() = 1 THEN
  IF fgl_wtkclient() = 1 THEN
    CALL fgl_winmessage ("Welcome from server to WTK",
                        "Pleased to meet you!!", "stop")
  ELSE
    CALL fgl_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. [Figure 7-3](#) shows the message that you see if you are using an X-Windows client.

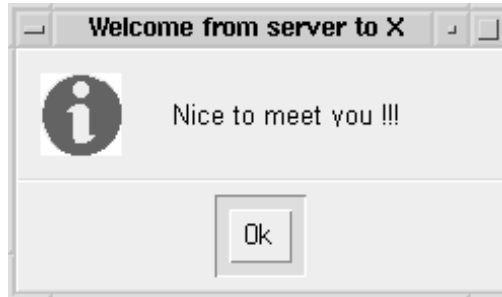


Figure 7-3
*Welcome Message
in Windows Client
Mode*

Window-Management Functions

This section describes the extensions that help you manage application windowing.

Setting the Default Size of a Window

The following function allows you to change the default size of the program window.

Syntax `fgl_setsize (nblines, nbcols)`

nblines Integer that specifies the new number of lines

nbcols Integer that specifies the new number of columns

Example:

```

...
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
...

```

Setting the Title of a Window

The following function allows you to set the title of a program window:

Syntax `fgl_settitle(mytitle)`

mytitle String or variable with the new title

The default title is the program name. To change this title, use the **fgl_settitle** function.

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 title field and then press ENTER. To quit this program, press the interrupt key.

Retrieving Information from a Field

The following function allows you to receive information about the currently prompted field during a dialog function:

Syntax call fgl_formfield_getoption ("option") returning var
 or
 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 = fgl_formfield_getoption("length")
    MESSAGE "No more than ",LGT," characters"
  END INPUT
...
```

Retrieving Information from an Application Window

The following function returns information about the current application window:

Syntax call fgl_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

<code>border</code>	Returns TRUE if the current window has a border; otherwise returns FALSE
<code>formline</code>	Returns the form line of the current window
<code>menuline</code>	Returns the menu line of the current window
<code>commentline</code>	Returns the comment line of the current window
<code>messageline</code>	Returns the message line of the current window
<code>errorline</code>	Returns the error line of the current window
<code>insertkey</code>	Returns the value of insertkey (value as with the fgl_getkey function)
<code>deletekey</code>	Returns the value of deletekey (value as with the fgl_getkey function)
<code>nextkey</code>	Returns the value of nextkey (value as with the fgl_getkey function)
<code>previouskey</code>	Returns the value of previouskey (value as with the fgl_getkey function)
<code>acceptkey</code>	Returns the value of acceptkey (value as with the fgl_getkey function)
<code>helpkey</code>	Returns the value of helpkey (value as with the fgl_getkey function)
<code>abortkey</code>	Returns the value of abortkey (value as with the fgl_getkey function)
<code>inputwrap</code>	Returns TRUE if the inputwrap option is on; otherwise returns FALSE
<code>fieldorder</code>	Returns TRUE if the fieldorder option is constraint; otherwise returns FALSE
<code>var</code>	The variable that contains the return value of the function

Example:

```
MAIN
DEFINE VAR CHAR(20)
CALL fgl_init4js()
OPEN WINDOW hello AT 2,2 WITH 20 ROWS, 50 COLUMNS
ATTRIBUTES(BORDER)
LET var = fgl_window_getoption("name")
DISPLAY "You are in window ",var AT 5,5
SLEEP 3
CLOSE WINDOW hello
END MAIN
```

Setting the Active Window

The following function makes the specified window, named *name*, the active window:

Syntax `fgl_window_current("name")`

name Specifies the name of a window

Example:

```
Call fgl_window_current("hello")
```

Closing a Window

The following function closes the window named *name*:

Syntax `fgl_window_close("name")`

name Specifies the name of a window

Example:

```
CALL fgl_window_close("name")
```

Creating Toolbars and Toolbar Icons

You can add a toolbar that contains icons that represents hot keys to the top of the screen. A corresponding help tip appears when the mouse pointer is positioned over an icon.

To enable tool bar functionality, add the following line to the **fglprofile** file:

```
gui.toolbar.visible = 1
```

To disable toolbar functionality, add the following line:

```
gui.toolbar.visible = 0
```

After this line, you might have groups of lines, with each group corresponding to an icon on the toolbar:

```
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.

Element	Description
<i>order</i>	The position of the icon in the toolbar.
<i>keytext</i>	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.
<i>bmpname</i>	Name of the bitmap file to use for the icon. The file <i>bmpname.bmp</i> must exist in directory \$FGLDIR/toolbars when using X11 or in directory \$WTKDIR/bmp of the Windows client when using Windows, where \$WTKDIR is the installation directory of the 4GL server.
<code>gui.toolbar.order.hideButton = {0 1}</code>	Indicates if the hot-key button corresponding to the icon must remain in the hot-key button frame in addition to the icon (hideButton = 0) or if it must disappear (hideButton =1).

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.

The following example shows a **fglprofile** configuration file:

```
gui.ToolBar.enabled = 1
gui.ToolBar.1.text = "OK"
gui.ToolBar.1.bmp = "exclam"
gui.ToolBar.1.hideButton = 1
gui.ToolBar.2.text = "Interrupt"
gui.ToolBar.2.bmp = "stop"
gui.ToolBar.2.hidebutton = 1
gui.ToolBar.3.text = "fglSeparator"
gui.ToolBar.3.bmp = "fglSeparator"
gui.ToolBar.4.text = "Help"
gui.ToolBar.4.bmp = "ques"
gui.ToolBar.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.

Creating Dialog Boxes

This section describes the extensions that affect dialog boxes.

Creating an Interactive Message Box

The following function displays an interactive box in a separate window with all possible answers in a menu:

Syntax `fgl_winbutton (title, text, default, buttons, icon, danger)`

title Title of the box

text Text of the question (\n stands for new line)

- default* Default button selected
- buttons* List of values separated by the pipe character (|)
- icon* Name of the icon to be used in the dialog box
Figure 7-4 shows possible message icon selections.

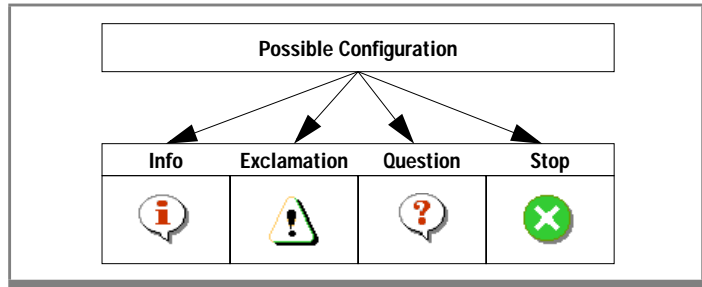


Figure 7-4
Message Icons

- danger* Number of the warning item: a skull with crossbones will appear each time the pointer enters the corresponding button (on X11 only)

You can put anything in the definition of a button, subject to the following rules:

- 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.

Displaying an Interactive Message Box

The following function opens a dialog box with all possible answers in a menu:

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 (|)
[Figure 7-5](#) shows sample message values.

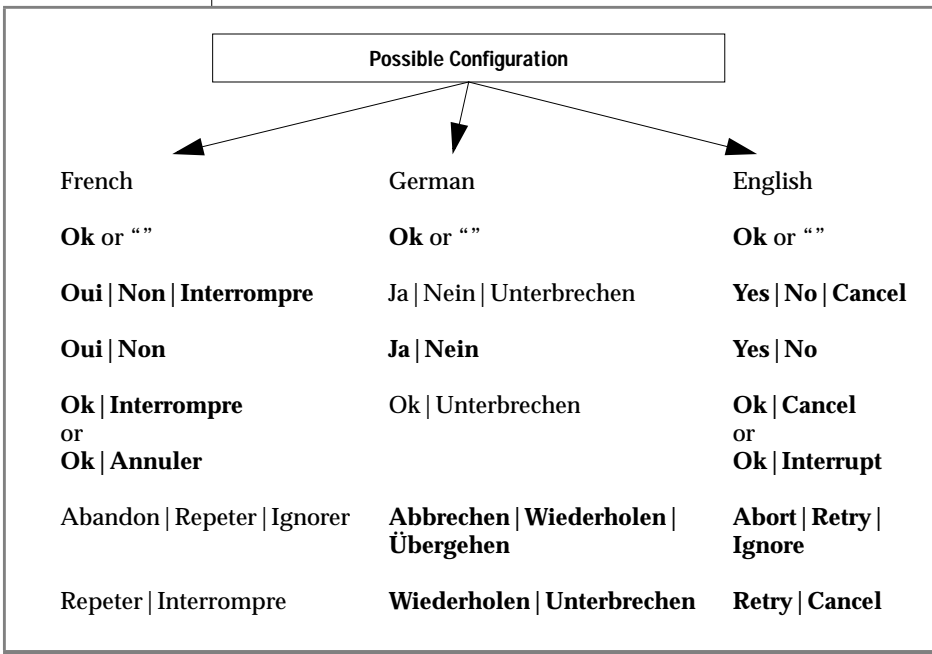


Figure 7-5
 Message Values

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

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 dialog box that [Figure 7-6](#) shows.

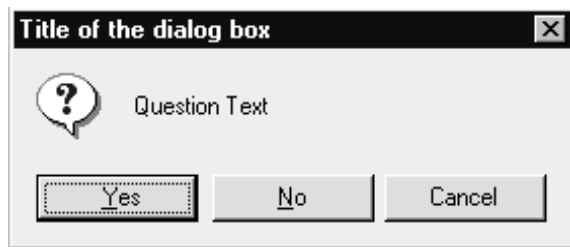


Figure 7-6
Dialog Box

This function replaces the typical PROMPT...FOR CHAR loop.

Formatting Text in a Message Box

The following 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

This function displays a message box with an OK button. For example:

```
MAIN
--#CALL fgl_init4js()
--#CALL fgl_winmessage("Title of the message", "Text or variable", "info")
END MAIN
```

This code produces the message box that [Figure 7-7](#) shows.

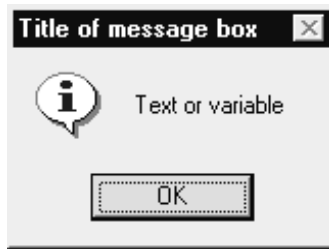


Figure 7-7
Message Box

Entering a Field Value into a Message Box

The following function displays a dialog box with a field that accepts a value:

Syntax `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

Returns Entered value

The following example shows 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

```

This code produces the dialog box that [Figure 7-8](#) shows.

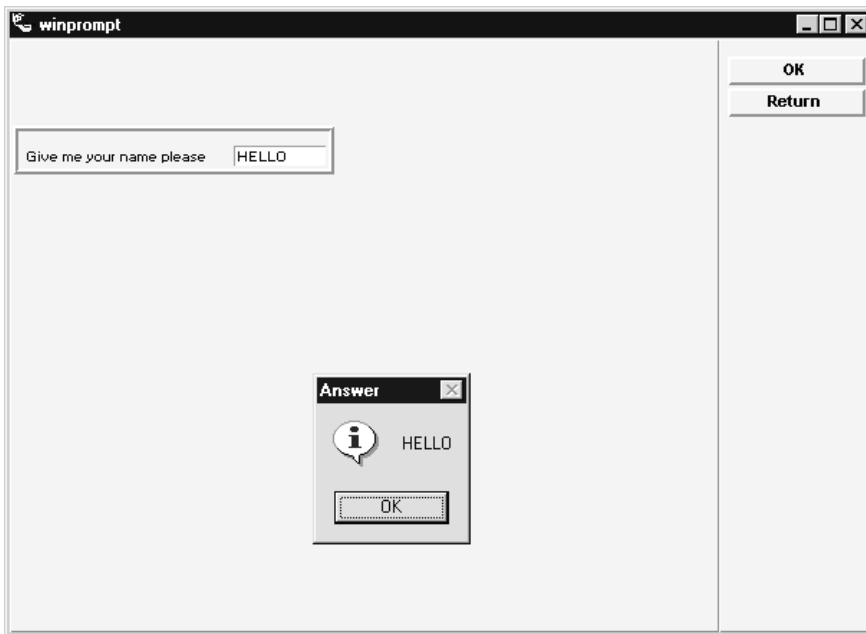


Figure 7-8
Dialog Box with
Entry Field

Using Drawing Extensions

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.

Mouse-Management Functions

Before you begin drawing, you might want to specify the behavior of your mouse. To manage mouse behavior, use the following functions.

Returning a Value After a Left Mouse Click

To define a key to be returned when you click the left mouse button, use the following function:

Syntax `drawbuttonleft(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 left mouse button.

Returns None

This function defines a key to be returned when you click the specified item with the left mouse button. For example:

```
CALL drawbuttonleft(num_item, "F4")
```

Returning a Value After a Right Mouse Click

To define a key to be returned when you click the right mouse button, use the following function:

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

For example:

```
CALL drawbuttonright(num_item, "Control-c")
```

Remove Key Binding

The following function removes all key binding on an item:

Syntax: `drawclearbutton(noit)`

noit INTEGER Item number returned by the function creating the object

Returns None

For example:

```
CALL drawclearbutton(num_item)
```

Defining the Drawing Area

The drawing area is defined in the same way as a screen array, as the following example shows.

In the file **draw2.per**:

```

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 ]
}
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).

Initializing the Drawing Function

The following 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 computer. If you call this function after you open the form that contains 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 daemon.

Selecting a Drawing Area

The following function selects an area in which to draw:

Syntax `drawselect(field_name)`

field_tag `CHAR(xx)` Field name in which you want to draw

Returns `None`

After a window that contains 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("draw")
```

Selecting a Drawing Color

The following function specifies the drawing color:

Syntax	<code>drawfillcolor(<i>color</i>)</code>	
<i>color</i>	CHAR(<i>xx</i>)	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 and in the **desi\etc** subdirectory of the Windows front-end installation directory. For example:

```
CALL drawfillcolor("red")
```

Specifying the Text Insertion Point

The following function specifies the insertion point for the text:

Syntax	<code>drawanchor(<i>pos</i>)</code>		
<i>pos</i>	CHAR(<i>x</i>)	<i>n</i>	Top of the text
		<i>e</i>	Right side
		<i>s</i>	Bottom side
		<i>w</i>	Left side
Returns	None		

Use **drawAnchor()** to specify the insertion point for the text before using the function **drawtext**. For example:

```
CALL drawanchor("n")
```

Changing Line Colors

The following function defines whether the color of the line can change:

Syntax	<code>disablecolorlines(<i>colorLines</i>)</code>		
<i>colorLines</i>	INTEGER	0	The lines take the color defined by DrawFillColor
		1	The lines are always black
Returns	None		

By default, the lines take the color defined by the **DrawFillColor** function.

Example:

```
CALL disablecolorlines(1)
```

Setting Line Width

The following function specifies the width of the line:

Syntax	<code>drawlinewidth(<i>width</i>)</code>	
<i>width</i>	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)
```

Clearing the Draw Function

The following function is the clear function:

Syntax	<code>drawclear()</code>
Returns	None

This function clears the drawing area specified by the **drawselect** function. For example:

```
CALL drawclear()
```

Drawing Rectangles

The following function draws a rectangle by specifying the lower left corner and the length:

Syntax	<code>drawrectangle(<i>y, x, dx, dy</i>)</code>	
<i>y, x</i>	INTEGER	Coordinate of the lower left corner
<i>dx, dy</i>	INTEGER	Length of the rectangle
Returns	The item number of the rectangle in the canvas	

Setting the Fill Color

The rectangle is filled with the color set using the function **drawfillcolor**. For example:

```
CALL drawrectangle(500,400,120,110) RETURNING ret
```

Drawing an Oval

The following function draws an oval:

Syntax	<code>drawoval(<i>y, x, dy, dx</i>)</code>	
<i>y, x</i>	INTEGER	Coordinate of the lower left corner
<i>dy, dx</i>	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:

```
CALL drawoval(500,400,150,100) RETURNING ret
```

Drawing a Circle

The following function draws a circle:

Syntax	<code>drawcircle(y,x,r)</code>
<i>y, x</i>	INTEGER The lower left corner of the bounding square that contains the circle
<i>r</i>	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:

```
CALL drawcircle(500,400,65) RETURNING ret
```

Drawing a Line

The following function draws a line:

Syntax	<code>drawline(y,x,dy,dx)</code>
<i>y, x</i>	INTEGER Coordinate of the first point of the line
<i>dy, dx</i>	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
```


Drawing Text

The following function draws text:

Syntax	<code>drawtext(<i>y</i>,<i>x</i>,<i>t</i>)</code>
<i>y</i> , <i>x</i>	INTEGER The starting point of the text
<i>t</i>	CHAR(<i>xx</i>) 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
```

Drawing an Arc

The following function draws an arc:

Syntax	<code>drawarc(<i>y</i>,<i>x</i>,<i>d</i>,<i>start</i>,<i>arc</i>)</code>
<i>y</i> , <i>x</i>	INTEGER The coordinate of the lower left corner
<i>d</i>	INTEGER The border length
<i>start</i>	INTEGER The start angle
<i>arc</i>	INTEGER The span of the arc
Returns	The item number of the arc in the canvas

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 **drawfillcolor**. For example:

```
CALL drawarc(500,400,100,12,25) RETURNING ret
```

Drawing a Polygon

The following function draws a polygon:

Syntax `drawpolygon(list)`

list `CHAR(xx)` List of coordinates

Returns The item number of the polygon in the canvas

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:

```
CALL drawpolygon("120 150 200 150 400 430") RETURNING ret
```

Configuring the Dynamic 4GL Compiler

In This Chapter	8-3
Configuring Dynamic 4GL	8-3
Runtime Configuration File	8-4
User Configuration File	8-4
Program Configuration File	8-4
General Configuration Settings.	8-5
Runtime Configuration Settings	8-6
General Settings	8-6
Graphical Daemon Autostart	8-9
UNIX Settings	8-10
Microsoft Windows Settings	8-11
License Configuration Settings.	8-14
General Settings	8-14
UNIX Settings	8-16
GUI Settings	8-17
General GUI Settings.	8-17
Menu GUI Settings	8-19
Status Bar Settings	8-36
Memory Mapping Settings	8-37
Local Editing Settings	8-38
Cut, Copy, and Paste Feature Settings	8-39

In This Chapter

This chapter describes all the settings available in the configuration file. For each setting, this chapter provides a description, possible values, and an example of the syntax. The configuration file has the following sections:

- Configuring Dynamic 4GL
- General configuration file settings
- Runtime configuration settings
- License configuration settings
- GUI settings
- Status Bar settings
- Memory mapping settings
- Local editing settings
- Cut, copy, and paste feature settings

Configuring Dynamic 4GL

You can control the behavior of the Dynamic 4GL compiler with the following three configuration files:

- Runtime configuration file
- User configuration file
- Program configuration file

Runtime Configuration File

The main configuration file, **fglprofile**, is located in the **\$FGLDIR/etc** directory. This configuration file is loaded first and is loaded each time you run an application.

User Configuration File

The user configuration file is specified by the **FGLPROFILE** environment variable. If this environment variable is set in the environment of the current user, the corresponding file is loaded after the **fglprofile** file. Entries defined in the two files are set to the value defined by the last loaded configuration file.

Program Configuration File

The program configuration file is located in the directory defined in one of the two previous configuration files with the entry named **fglrun.default**. (It is usually in the **\$FGLDIR/defaults** directory.) Use this configuration file to control program-specific behavior.

General Configuration Settings

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.

Do not change this value.

Value Resource filename

Default `fgl2c.res`

Syntax `fglrun.interface="fgl2c.res"`

fglrun.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 `fglrun.scriptname="fgl2c.tcl"`

fglrun.defaults

Description	Specifies in which directory the program-specific configuration files will be searched.
Value	Complete path to the specific configuration files
Default	<code>\$FGLDIR/defaults</code>
Syntax	<code>fglrun.defaults="\$FGLDIR/defaults/"</code>

Runtime Configuration Settings

This section describes the settings that affect runtime configuration.

General Settings

This section describes the general settings for the runtime configuration section of the configuration file.

fglrun.arrayIgnoreRangeError

Description	Ignores range control in arrays. If this variable is set to 1, if <code>x</code> is an array, <code>x[-1]</code> gives no error but NULL. If this variable is set to 0, <code>x[-1]</code> gives error -1326.
Value	0 or 1
Default	0
Syntax	<code>fglrun.arrayIgnoreRangeError=1</code>
Recommendation	Set to 1

dialog.fieldOrder

Description	Determines whether the intermediate event triggers (AFTER/BEFORE 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	<code>dialog.fieldorder=0</code>

report.aggregateZero

Description	Determines the value to be returned by report aggregate functions (<code>avg</code> , <code>sum</code> , ...) when the result is NULL.
Value	0 returns NULL 1 returns ZERO
Default	0
Syntax	<code>report.aggregateZero=0</code>

gui.chartable

Description	Defines a conversion file to be used for characters under GUI. It will be searched in the <code>\$FGLDIR/etc/</code> directory. You can create a file with the <code>mkchartab</code> utility (see Appendix D).
Value	The path from the <code>\$FGLDIR</code> directory to the filter file
Default	None
Syntax	<code>gui.chartable="iso/ansinogr"</code>

fglrun.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	<code>fglrun.cursor.global=0</code>

fglrun.ix6

Description	Commands the P-code runner (fglrun) to act like INFORMIX-4GL 6.x. For more information, see Chapter 6, “Using Form Extensions to 4GL.”
Value	0 to react like INFORMIX-4GL 4.x 1 to react like INFORMIX-4GL 6.x
Default	0
Syntax	<code>fglrun.ix6=0</code>

fglrun.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	<code>fglrun.warning.logfile=0</code>

Graphical Daemon Autostart

This section describes the settings that control the startup of the graphical daemon.

fglrun.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"`

fglrun.server.number

Description Specifies the maximum number of graphical servers to autostart.

Default 100

Syntax `fglrun.server.number=50`

fglrun.server.x

Description With X11, Citrix Winframe, and Microsoft Terminal Server client computers, 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 None

Syntax `fglrun.server.1="client:0, client:0.0"`

UNIX Settings

This section describes the settings specific to UNIX configurations.

fglrun.signalOOB

Description To send the interrupt signal to the server from the client, OOB data is sent over the network. On some UNIX systems, the number of the OOB data might 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`

Microsoft Windows Settings

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 Windows NT.

A trailing space is required after the command.

Value Name of the command to execute

Default `cmd /c`

Syntax `fglrun.cmd.winnt="cmd /c "`

fglrun.cmd.win95

Description Shell command to perform the RUN WITHOUT WAITING command on Windows 95.

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 Windows NT, it will export the following two environment variables to the database:

INFORMIXSERVICE (default **turbo**)
INFORMIXPROTOCOL (default **olsocTCP**)

Value Name of the variable to distinguish remote connection from network drive

Default REMOTEADDRESS

Syntax fglrun.remote.envvar="REMOTEADDRESS"

fglrun.database.listvar

Description This variable must be set on Windows NT computers using Informix 7.2x database servers. It contains the list of all Informix variables. On Windows 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 COLLCHAR CONRETRY CONTIME DBANSIWARN
DBDATE DBLANG DBMONEY DBNLS DBPATH DBTEMP DBTIME
DELIMIDENT 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
DBMONEYSIZE DBSS2 DBSS3"

nt.withoutoob

Description	Determines if your Windows 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	<code>nt.withoutoob=0</code>

fglrun.setenv.o

Description	Sets an environment variable to a specific value even if the variable is already defined. For each environment variable, increment the value of <i>x</i> by 1, to create distinct resource names.
Value	Name and value of the environment variable to set
Default	None
Syntax	<code>fglrun.setenv.0="INFORMIXDIR=c:\informix"</code>

fglrun.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 computer on which the Informix database server runs) to use a remote session on Windows NT. Increment the value of <i>x</i> by 1 to create distinct resource names.
Value	Name and value of the environment variable to set
Default	None
Syntax	<code>fglrun.defaultenv.0="INFORMIXSQLHOSTS=\\IXSERVER"</code>

License Configuration Settings

This section describes the settings that affect licensing.

General Settings

This section describes the general settings for the license configuration section of the configuration file.

fgllic.server

Description Name of the computer 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"`

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	<code>fgllic.local=0</code>

fgllic.ping

Description	Time limit for the ping to detect the license server computer 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	<code>fgllic.ping=5000</code>

UNIX Settings

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"`

fgllic.ps

Description Command giving the complete process list for a computer.

Value Command name and flag for listing all processes that run on a computer

Default `ps -ae`

Syntax `fgllic.ps="ps -ae"`

GUI Settings

This section describes the settings that affect configuration of the GUI.

General GUI Settings

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 that indicates 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 computer (especially Windows computers) does not support the OOB mechanism you must disable it. In this case, a second, slightly more time-consuming method 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 Settings

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 Settings

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 computers, where the bitmap files that the toolbar uses 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.visible

Description Enables the toolbar in your program.

Value 0 Disables the toolbar
1 Enables the toolbar

Default 0

Syntax `gui.toolBar.visible = 0`

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 that contains 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	<code>gui.toolbar.0.hideButton = 0</code>

gui.toolbar.0.key

Description	Name of the key used with this icon. This variable can be used instead of gui.toolbar.0.text .
Value	Key name associated with the toolbar
Default	None
Syntax	<code>gui.toolbar.0.key = "F1"</code>

gui.toolbar.0.text

Description	Text associated with an icon on the toolbar. For menu command text, this is only available with vertical menus.
Value	Text associated with the icon
Default	None
Syntax	<code>gui.toolbar.0.text = "Help"</code>

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`

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 Settings

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`

gui.screen.x

Description X position of an application window.

Value `incr` An incremented position (In this case, you have to set **gui.screen.incrx**).

`center` Centers the window in the screen

`number` An absolute position, in characters

Default `incr`

Syntax `gui.screen.x = "incr"`

gui.screen.incrx

Description Specifies the increment for the display of the application windows (in number characters) on the horizontal axis.

Value Number of characters

Default `3`

Syntax `gui.screen.incrx = 3`

gui.screen.y

Description Y position of an application window.

Value `incr` An incremented position (In this case, you have to set **gui.screen.incry**).

`center` Centers the window in the screen

`number` An absolute position, in characters

Default: `incr`

Syntax `gui.screen.y = "incr"`

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 0 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.

1 Normal mode

Default 1

Syntax `gui.screen.withwm = 0`

Key GUI Settings

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.

Value Offset for the key code SHIFT-F1

Default 12

Syntax `gui.key.add_function=12`

gui.key.interrupt

Description Specifies the name of the interrupt key.

Value Name of the interrupt key

Default "Control-c"

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.

Value Name of the key code to be sent to the program

Default "KEY_accept"

Syntax `gui.key.doubleClick.left = "F30"`

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 None

Syntax `gui.key.0.translate = "KP_Decimal comma"`

Using the Key.tcl Script

This file is located in the **\$FGLDIR/etc** directory. This script allows you to test the value mapped to a key.

For the X11 Client, start the **\$FGLDIR/etc/key.tcl** script with the following UNIX statement:

```
$ owish -f key.tcl
```

For the Windows client:

1. Copy key.tcl script to your local drive.
2. Create a new icon.

For instance, copy the WTK server and edit the icon properties.

3. In the Command line, add the following:

```
c:\fgl2cusr\bin\wtk.exe -d -f <path_name>\key.tcl
```

where **<path_name>** is the path to **key.tcl** on your local drive.

After you start the key.tcl script, type the key or key combinations. Their ASCII value and name are displayed in the debug window or in the terminal.

For example:

```
owish -f $FGLDIR/etc/key.tcl
Control_L 66
Control-c 56
Up 91
Down 92
Right 97
Left 87
Delete 84
KP_Enter 116
KP_3 111
KP_Decimal 112
Shift_R 65
Shift-exclam 10
```

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**

The following table lists keys for specific actions.

Key	Description
key.help.text	Text for the help key
key.accept.text	Text for the accept key
key.interrupt.text	Text for the interrupt key
key.delete.text	Text for the delete key

Key	Description
key.insert.text	Text for the insert key
key.return.text	Text for the return key
key.escape.text	Text for the escape key
key.prevpager.text	Text for the previous page key
key.nextpage.text	Text for the next page key

The following table lists the function keys.

key.f1.text = F1	key.f13.text = F13
key.f2.text = F2	key.f14.text = F14
key.f3.text = F3	key.f15.text = F15
key.f4.text = F4	key.f16.text = F16
key.f5.text = F5	key.f17.text = F17
key.f6.text = F6	key.f18.text = F18
key.f7.text = F7	key.f19.text = F19
key.f8.text = F8	key.f20.text = F20
key.f9.text = F9	key.f21.text = F21
key.f10.text = F10	key.f22.text = F22
key.f11.text = F11	key.f23.text = F23
key.f12.text = F12	key.f24.text = F24

The following table lists the **Control modified** keys.

key.control-a.text = Control-a	key.control-n.text = Control-n
key.control-b.text = Control-b	key.control-o.text = Control-o
key.control-c.text = Control-c	key.control-p.text = Control-p
key.control-d.text = Control-d	key.control-q.text = Control-q
key.control-e.text = Control-e	key.control-r.text = Control-r
key.control-f.text = Control-f	key.control-s.text = Control-s
key.control-g.text = Control-g	key.control-t.text = Control-t
key.control-h.text = Control-h	key.control-u.text = Control-u
key.control-i.text = Control-i	key.control-v.text = Control-v
key.control-j.text = Control-j	key.control-w.text = Control-w
key.control-k.text = Control-k	key.control-x.text = Control-x
key.control-l.text = Control-l	key.control-y.text = Control-y
key.control-m.text = Control-m	key.control-z.text = Control-z

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`

"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 Settings

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.

Value 0 Use X11 style
 1 Use Windows style

Default 0

Syntax `gui.mswindow.scrollbar=0`

gui.user.font.choice

Description Restricts the end user from changing the fonts of the application with the Windows front-end menu at runtime.

Value 0 The user is able to change the fonts
 1 The user cannot change the fonts (except by changing the **local.tcl** file)

Default 1

Syntax `gui.user.font.choice=1`

Status Bar Settings

In a graphical client, the state of special keys can be displayed on the Status Bar, including: Caps Lock, Num Lock, and Scroll Lock. For each key, you can configure the text that appears on the Status Bar.

gui.statusBar.indicator.x.source

Description Identifies the keys whose status appears on the status bar.

Value `gui.statusBar.indicator.x.source="capsLock | numLock | scrollLock"` where `x` is a number between 1 and 3. Values are not case sensitive.

Default `gui.statusBar.indicator.1.source = "capsLock"`
`gui.statusBar.indicator.2.source = "numLock"`

Syntax `gui.statusBar.indicator.3.source = "scrollLock"`

gui.statusBar.capsLock.text

Description Text that appears on the Status Bar for the capsLock key.

Value Any text value.

Default "CAPS"

Syntax `gui.statusBar.capsLock.text = "MAJ"`

gui.statusBar.numLock.text

Description Text that appears on the Status Bar for the numLock key.

Value Any text value.

Default "NUM"

Syntax `gui.statusBar.numLock.text = "NUM"`

gui.statusBar.scrollLock.text

Description Text that appears on the Status Bar for the scrollLock key.

Value Any text value.

Default "SCROLL"

Syntax `gui.statusBar.scrollLock.text = "SCROLL"`

Memory Mapping Settings

Memory mapping allows the runtime system to load a single version of a P-code application and share it across multiple connections, significantly reducing the memory required on the application server.

The **fglmkrun** shell script output tells you if the runners have been built with or without the memory mapping emulation.

***Tip:** Some systems do not support memory mapping. In this instance, an emulation of this feature is provided. For Windows NT, an emulation is always used.*



fglrun.mmapDisable

Description Enables or disables memory mapping.

Value 1 Disable memory mapping

0 Enable memory mapping

Default 1

Syntax `fglrun.mmapDisable = 1`

Local Editing Settings

The local editing feature reduces the communication between the server (where the application is running) and the client (where the application appears). Enabling this feature can reduce the network traffic and might speed up applications.

gui.local.edit

Description Enables or disables local editing.

Value 1 Enable local editing

0 Disable local editing

Default 1

Syntax `gui.local.edit = 1`

Cut, Copy, and Paste Feature Settings

The cut, copy, and paste feature allows you to cut or copy a selected string from one field to another in a 4GL graphical application. To use this feature, the local editing feature should be enabled.

To enable this feature, edit the **fglprofile** file. The local editing feature should be enabled with:

```
gui.local.edit = 1
```

Then you can choose the short cut keys for cut, copy, and paste with:

```
gui.key.copy = "Control-c" (default value is Control-Insert)
gui.key.paste= "Control-v" (default value is Shift-Insert)
gui.key.cut= "Control-x" (default value is Delete)
```



Warning: If you want to redefine *gui.key.copy* as "Control-C", set *gui.key.interrupt* to another value.

You can also define the message displayed when the user tries to use a local editing feature in a not allowed field with:

```
gui.local.edit.error = "error string"
```

You can use the following keyboard equivalents:

- key Shift-Left: add character at the left to the selection
- key Shift-Right: add character at the right to the selection
- key Shift-Home: add from current character to first character to the selection
- key Shift-End: add from current character to last character to the selection

Example

The following code sample creates a 4GL application with two fields.

File **demo.per**:

```
database formonly
screen {

[f01                ]

[f02                ]

}
attributes
f01=formonly.f01;
f02=formonly.f02;

File demo.4gl:
MAIN
DEFINE f01, f02 CHAR(20)
OPEN WINDOW w1 AT 1,1 WITH FORM "demo"
MENU "Cut&Paste"
  COMMAND "Input"
    INPUT BY NAME f01, f02
  COMMAND "Exit"
    EXIT MENU
END MENU
END MAIN
```

Then compile and run this program.

With the Windows client you can:

- Type text in the first field, select the text with the mouse or the shift key plus any of the left, right, home, or end keys. Then copy or cut the string with the assigned keys in the **fglprofile**. You can paste the string into the second field or another Windows application.
- Type text in one of the fields, select a few characters in this field and type new characters. The new characters replace the whole selected string.

With the X11 client, you can copy and paste between applications running the same graphical daemon (that is, applications running with the same FGLSERVER value) exactly like with the Windows client. However, to copy a string to another X11 application (that is any Dynamic 4GL application running with a different FGLSERVER value), you have to perform the following steps:

- Select the string you want to copy with the mouse
- Select the 4GL application where you want to paste the string
- Press the Copy key
- Select the place where you want to paste the string and press the Paste key.

The reason for this is that X11 does not offer a Windows-type clipboard. Instead, Dynamic 4GL implements something similar to a Windows Clipboard for each application running the same FGLSERVER value. When you paste a string into a 4GL application, you need to put the string in the corresponding clipboard for the application.

Using the Configuration Manager

In This Chapter	9-3
About the Configuration Manager	9-3
Starting the Configuration Manager	9-3
Starting on UNIX	9-4
Starting on Windows	9-4
Using the Dynamic 4GL Configuration Manager	9-4
File Menu	9-5
Widget Menu	9-5
Label Object	9-5
Attributes Object	9-6
Colors Object	9-6
Button Object	9-7
Field Object	9-8
Scrollbar Object	9-8
4GL-Windows Object	9-8
The Help Tip Object	9-9
The Help Menu	9-9
How to Configure an Object with the Configuration Manager	9-10
Opening a File	9-10
Configuration Types	9-10
Color Choice	9-11
Radio Button Choice	9-11
Numeric Field	9-11
The Different Configurations	9-12
Color Configuration	9-12
Relief Configuration	9-13
Border Width Configuration	9-13
Relief and Border Width Attributes	9-13
Attribute for a Specific Window	9-14

In This Chapter

This chapter describes how to set properties for GUI controls on both UNIX and Windows.

About the Configuration Manager

The Configuration Manager allows you to set the properties for any GUI controls. It does this by updating a configuration file. By default, the configuration file is called `$HOME/.fgl2crc` on UNIX and `locals.tcl` on Windows.

The Configuration Manager also lets you manage these configuration files and configure different graphical widgets. For instance, you can configure:

- foreground and background colors
- buttons
- radio buttons
- fields

Starting the Configuration Manager

You can use the Configuration Manager on UNIX or Windows NT. You should start the Configuration Manager on the client side. This means that if the compiler is installed on Unix, and you are using Windows clients (WTK), you should start the configuration on Windows.

Starting 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 the Configuration Manager, you must be in graphical mode (**FGLGUI = 1**), and the **DISPLAY** environment variable must be set. Then enter the **confdesi**, command at the shell prompt:

```
$ confdesi
```

Starting on Windows

The Configuration Manager is installed with the Windows client. The configuration file 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 **Informix-Config-Tools** icon.

Using the Dynamic 4GL Configuration Manager

The program interface contains the following three menus:

- **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 the following four items:

- | | |
|----------------|--|
| Open | Opens an existing configuration file. By default, the Configuration Manager offers the standard filename as the default, depending on the operating system (<code>\$HOME/.fgl2crc</code> on UNIX and <code>locals.tcl</code> on Windows). |
| Save | Saves changes to the configuration file using the default name. The updated configuration file overwrites the old file. |
| Save to | Saves the configuration file using the specified filename. The default depends on the operating system. You can change it to another filename. |
| Exit | Exits 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.

Label Object

The **Label** object contains the configuration of the **Label** item:

- | | |
|----------------|--|
| Label | Any label that is not generated, such as the message label. Only the border width can be configured. |
| Message | A label generated with a <code>DISPLAY AT</code> , <code>MESSAGE</code> , or <code>COMMENT</code> statement. |
| Error | A label generated with the <code>ERROR</code> 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.

Attributes Object

The **Attributes** object allows you to configure the attributes used in the DISPLAY {AT | TO}, INPUT, CONSTRUCT, and PROMPT statements.

All Attributes Select all the attribute objects.

Individual Attribute Select each attribute object individually.

You can specify a different configuration for each combination of attributes. The relief option can only be applied to attribute combinations using BLINK.

Colors Object

The **Colors** object allows you to configure the standard eight colors (white, black, yellow, magenta, red, cyan, green, and blue) that INFORMIX-4GL uses. You assign a specific color to the standard colors.

Foreground Color of characters and lines.

Background Color for windows, toolbars, and entries.

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.

Button Object

The **Button** object allows you to configure 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 Title Menu** Button that contains 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, 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.
- Radiobutton** 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.
- Checkbutton** 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.

You can use two kinds of buttons: TK buttons or Windows buttons. Your choice depends on the value of the **gui.mswindow.button** entry point in the file **\$FGGLDIR/etc/fglprofile**. For more information about the **fglprofile**, see [Chapter 5, “Using Non-Graphical Extensions to 4GL,”](#) and [Appendix B, “Common Problems and Workarounds.”](#)

Field Object

The **Field** object allows you to configure 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. |

Scrollbar Object

The **Scrollbar** object allows you to configure the scrollbar used with the screen record. For this object, foreground, background, and active foreground can be configured.

4GL-Windows Object

This object allows you to configure 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.
- 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 might not see this color if you do not have enough space between the buttons.

The Help Tip Object

The **Help tip** object allows you to configure 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 with the Configuration Manager

The following sections describe how to use the Configuration Manager.

Opening a File

1. Choose the **Open** or **New** commands from the File menu commands to open an existing configuration file or create a new configuration file.
2. Choose an object from the **Widget** menu.
For more information on the different objects, see [“Widget Menu” on page 9-5](#).
3. Position the cursor on the object and click the right mouse button to display a menu of configuration options.
4. Double-click a menu option to configure the object.

Configuration Types

This section describes the configuration types that control:

- Color choice
- Radio button styles
- Numeric fields

Color Choice

The color choice allows you to choose a color from a palette; 20 to 40 colors appear at the bottom of the Configuration Window. This type is used by the following properties:

- **Background**
- **Foreground**
- **Active Bg**
- **Active Fg**
- **HighlightBg**

Click the **Prev** and **Next** buttons to display additional color sets. Configuration can 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.

Radio Button Choice

The radio button choice allows you to choose from among a set of values. Clicking an option applies the chosen configuration to the displayed item. This type is used by the following properties:

- **Borderwidth**
- **Relief**

Confirm your selections by clicking **Done** (or **Cancel** to cancel any modifications).

Numeric Field

Enter a numeric value for this configuration type. This type is used by the **Height** property.

Click **Apply** to show the effect of the value entered on the item displayed. After you enter a value, click **Done** or **Cancel**.

The Different Configurations

The following section describes different configurations.

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
Done	Accepts the new color configuration
Cancel	Aborts the color configuration
Old	Shows the previous configuration

You can specify the following color property settings:

Background	Specifies the color of the normal background of an item. This configuration is applied to all items.
Foreground	Specifies the color of the normal foreground of an item. This configuration is applied to Scrollbar and Color objects.
Active Bg	Specifies the background color choice of an item when it is active. For example, when the pointer is positioned on a button or a field when it is accessible for input. This configuration is applied to Button and Field objects.
Active Fg	Specifies 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 Scrollbar and Field objects.
HighlightBg	Specifies 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

Specifies the item relief style: raised, sunken, flat, grooved, or ridged. This configuration is applied to **Button**, **Field**, and **Screen** objects.

Border Width Configuration

Specifies the 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.

Relief and Border Width Attributes

For every combination of the attributes **BOLD**, **REVERSE**, **UNDERLINE**, and **BLINK**, a different relief, border width, and background color can 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.

The **Attributes** option can be useful for hiding specific input fields on the screen:

- With the ASCII interface, you can hide input fields by using the following setting in the **.per** file:

```
DELIMITER= " ";
```

- With the GUI, you can hide input fields by defining an attribute (**BLINK** for instance) with flat relief and the same color as the screen background. To hide the field, use the following command:

```
DISPLAY TO fieldname ATTRIBUTE(BLINK)xx
```

Attribute for a Specific Window

You can define background, relief, and color independently for all **Screen** objects by means of the menu option **Window**.

Using the HTML Client

In This Chapter	10-5
Web Deployment Architecture	10-6
Why Deploy on the Web?	10-7
HTML Client Limitations	10-8
HTML Client Enhancements	10-9
Installing the HTML Client	10-9
Installing on UNIX	10-9
Web Deployment Component Requirements	10-10
Components on the CD	10-10
Automatic Installation	10-10
Installing on Windows NT	10-13
Web Deployment Component Requirements	10-13
Location of Web Deployment Components	10-14
Running the Installation Program	10-14
Configuring Your System	10-15
How Web Deployment Works at Runtime	10-16
Supplying Your Own Headers and Footers	10-19
Disabling Password Display	10-19
Similarities Between a .per File and an .html File	10-19
Deploying a Sample Application	10-20
Screens.	10-22
Step 1: Creating a Dynamic 4GL Application	10-22
Step 2: Editing the Server Configuration File.	10-23
Examples of Configuration Settings	10-23
Results of Updating the Application Configuration File	10-29
Step 3: Creating a Script to Initialize the Application	10-31
Step 4: Editing Your Client Configuration File	10-31

Step 5: Starting the HTML Server Process on UNIX	10-31
Step 6: Starting the Browser	10-32
Step 7: Using the Application	10-32
Step 8: Enhancing the Application	10-36
Creating Email and Web Site Links	10-36
Enhancing the Screen Files	10-37
Horizontal split	10-39
Table	10-39
How Links Between Pages Work	10-40
HTML Emulation for Tables	10-41
Dynamic 4GL Features	10-41
Security Levels	10-44
Default Security.	10-44
Recommendations for Enhancing Security	10-46
SSL.	10-46
Using a Filtering Router	10-46
Using a Firewall	10-46
Application, Web Server, and Database Security	10-46
Certificate Authority.	10-47
Preventing Security Problems	10-47
Configuring the Web Deployment Software	10-48
Configuration Settings in the fgld.conf file.	10-48
Location	10-48
fglserver	10-49
debug	10-49
HTMLdebug.	10-50
Security	10-50
Apache Web Server	10-50
Microsoft IIS/Personal Web Server 4.0	10-50
Security Through the Web Server.	10-51
Security Through the File System.	10-51
Summary	10-52

Configuring the appname.conf File10-52
General Configuration Settings10-52
Version10-53
Application Name10-53
Client10-53
Service10-54
Server Number10-54
Security Level10-54
Time Out10-55
Maximum Tasks10-55
Debug10-55
Pre and Post Messages10-56
Header10-56
Footer10-56
Error10-57
Time-Out Message10-57
Too Many Tasks10-58
Normal Termination10-59
Styles10-59
Button Down10-60
Error Down10-60
Menu as Link10-60
Button Width10-60
Menu Button Width10-61
Emulate HTML10-61
Image Path10-61
Image Alternate Text10-62
Image Border10-62
Spawning10-63
Spawning Method10-63
Program10-63
Runner Name10-64
Runner Target10-64
Runner Environment10-65
Arrays10-65
Array as Button10-66
Array Image10-66
Troubleshooting the UNIX Installation.10-66
Checking the HTML Client10-66
Checking the HTML Server10-68

Manual Installation on UNIX	10-69
Extracting the Files	10-69
Installing the HTML Client on the Web Server	10-70
Installing the HTML Server on the Application Server	10-71
Installing the HTML Documentation on the Web Server	10-72
Installing the Example	10-72
Troubleshooting the Windows NT Installation	10-73
Checking the HTML Client	10-73
Checking the HTML Server.	10-73

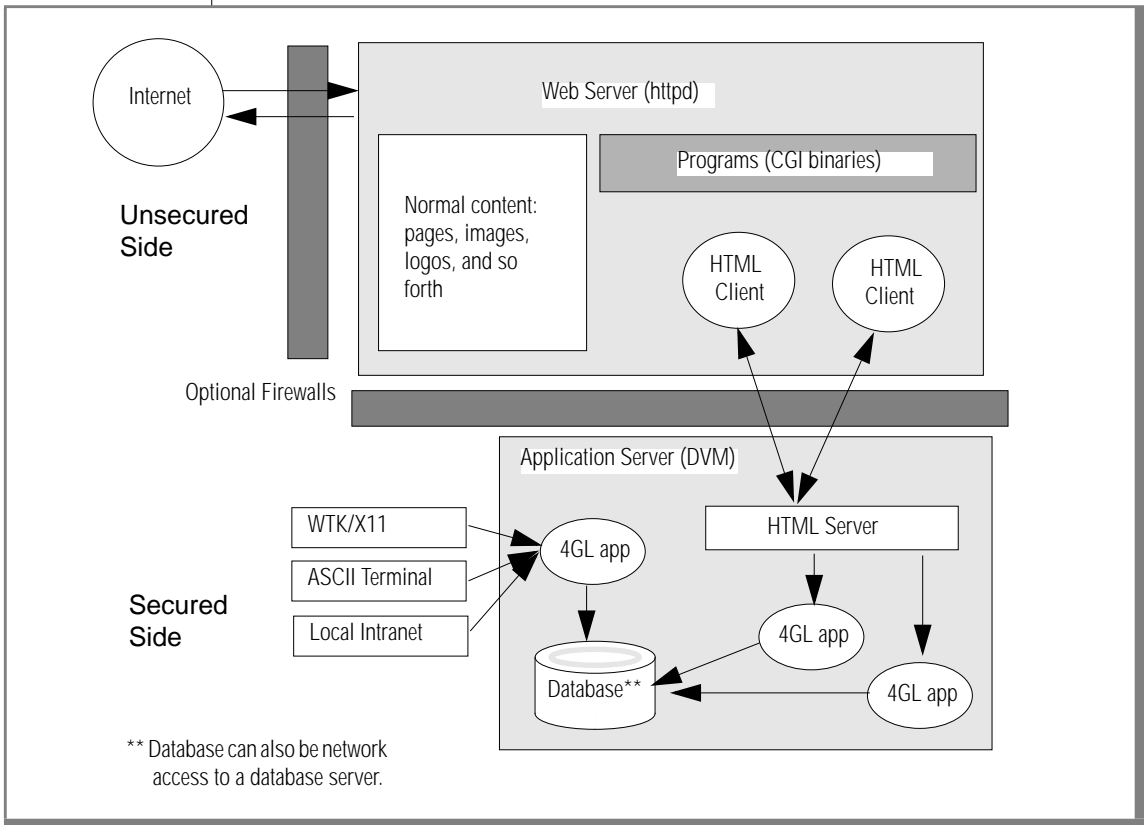
In This Chapter

This chapter describes how to deploy your Dynamic 4GL applications on a Web server. In addition, you can enhance the appearance of the Dynamic 4GL application for display with a Web browser.

Web Deployment Architecture

Figure 10-1 shows an overview of the Web deployment architecture.

Figure 10-1
Web Deployment Architecture



The components shown in the figure are as follows:

1. **Application Server:** The server that runs the Dynamic 4GL program. The term *application server* can refer either to a physical computer or to the software process (fglrun) that runs the application.
2. **Web Server:** The server (httpd) that makes Web pages available to browsers.

3. **HTML Client:** The HTML Client needs to run on the same computer as the Web server.
4. **WTK/X11:** The Windows and X11 clients.
5. **ASCII terminal:** The ASCII client.
6. **HTML Server:** The HTML server (**fglhtml**) typically runs on the application server computer.



***Important:** While the entire architecture can exist on one computer, this is not a typical Web deployment architecture.*

Why Deploy on the Web?

Deploying your Dynamic 4GL applications on the Web offers the following benefits:

- No need for special programming
You can deploy your existing Dynamic 4GL applications on the Internet or on an Intranet.
- Preserve application appearance and functionality
You can run your Dynamic 4GL application on the Web as easily as you can run it on your local computer.
- Ability to configure the user interface
You can customize your application's user interface to optimize it for display on the Web.
- Consistent application and database development
You can create applications for deployment in different environments using only one tool; you can reuse your existing application logic; you can use the same database for all the environments in which you deploy your application.
- Security
You can take advantage of options such as firewalls and secure socket layer (SSL) in addition to preserving the security features in your original application.

HTML Client Limitations

With Dynamic 4GL applications deployed on the Web, a Web limitation exists that each form must be transmitted to the server. In addition, some limitations are due to differences between 4GL and Dynamic 4GL, including:

- With your ASCII 4GL, each character is analyzed as it is typed.
- With local editing with WTK or X11, in Dynamic 4GL, each field is analyzed on the display server and transmitted to the runner when the field is completed.

When using the HTML client, you can expect the following effects:

- Only the top-level window is visible and it appears as one HTML page. Opening a submenu creates a new page, and only the information displayed on the submenu is visible to the user.
- The ON KEY actions do not work during input statements because the Dynamic 4GL program does not see the individual keystrokes. For the same reason, the COMMAND KEY options in menus also do not work.
- The BEFORE FIELD and AFTER FIELD clauses have no effect on the input sequence. The Dynamic 4GL program receives all data after it has been entered. This also means that NEXT FIELD has no effect, and that displaying the results of lookup data has no effect, either. You can use the BEFORE FIELD and AFTER FIELD clauses to validate the entered data, but you have limits on what you can do when the data is incorrect.
- The SLEEP statement without interaction has no effect.
- The PROMPT statement has no effect.

- The `DISPLAY`, `MESSAGE`, and `ERROR` statements only take effect when the user interacts with the program, such as when an `INPUT`, `INPUT ARRAY`, `DISPLAY ARRAY`, `CONSTRUCT`, or `MENU` statement is used. `RUN WITHOUT WAITING` can be used, but it cannot be used to start a new Dynamic 4GL program that interacts with the user through the Web browser. The user must decide to access the new program.
- Forward and Back browser buttons do not work reliably. When using the Forward and Back buttons, the HTML client has to go back to the server for information. Unfortunately, Dynamic 4GL applications are usually not written to go back to a screen form (unless you specifically accounted for this behavior when you wrote the Dynamic 4GL code).

HTML Client Enhancements

You can add the following enhancements to your HTML client applications:

- Customized page headers and footers HTML tags in the screen portion of the form HTML tags as labels
- HTML tags in your 4GL modules

Installing the HTML Client

The following two sections show how you can install components to deploy your Dynamic 4GL applications on a Web server for UNIX and Windows NT.

Installing on UNIX

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

Web Deployment Component Requirements

To install this software, you need to have installed (and have running):

- Dynamic 4GL compiler with a valid license number
- Web (HTTP) server
- 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.

Components on the CD

After 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 Bourne installation script, **`install.sh`**.

Automatic Installation

You can begin the automatic installation by executing either the self-extracting package or the Bourne shell script.

Using the Self-Extracting Package

To use the self-extracting package and install all the Web deployment components, enter:

```
sh html-all.sh -i
```

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

To use the shell script, you need the UNIX **gzip** and **tar** utilities.

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 -xf -
```

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

When you install the complete package, you are prompted 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.
- 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 **\$FGLDIR/cli-html/example**.
- Whether you want to install the release notes
- The path to the directory in which to install the release notes
The default is **\$FGLDIR/cli-html/release**.

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 [“Configuration Settings in the fglcl.conf file” on page 10-48](#).

Installing on Windows NT

This section provides directions to install the Web Deployment Components on Windows NT.

Web Deployment Component Requirements

To install this software, you need to have installed (and have running):

- Dynamic 4GL compiler with a valid license number
- Web (HTTPD) server
- 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.

Location of Web Deployment Components

After 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 install the HTML client software on Windows NT, execute the file named **cli-html.exe**.

During the installation, you are prompted to supply the following information. For each prompt, respond and click **Next** to continue the installation.

- A location for the HTML client (**Choose Destination Location** screen)
The default is **C:\I4glstrv\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 manually copy 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 10-70](#).

Available Packages

Customized installation lets you select any of the following packages:

- **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 10-72](#). The information applies equally to UNIX and Windows NT.

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 [“Configuration Settings in the fglcl.conf file” on page 10-48](#).

How Web Deployment Works at Runtime

Figure 10-2 shows an overview of the process followed by the HTML client, HTML server, and Web server as your application starts.

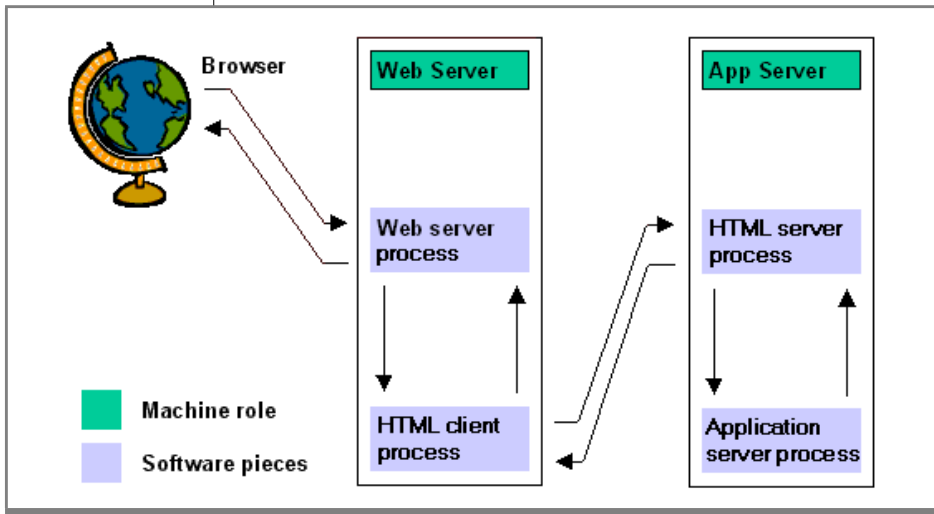


Figure 10-2
Web Deployment
Runtime Process

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

- The *application server* runs the main Dynamic 4GL program.
- The *HTML client*, **fglcl**, handles communication with the HTML server.
- The *HTML server*, **fglhtmlsd**, 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.

Important: This description omits network security details.



The steps are as follows:

1. The **fglhtml** program (HTML server process) starts and listens to the TCP/IP ports that you defined in the **fglcl.conf** configuration file. The default port value is 6500.

This action can be done manually (for instance, on a development system) or can be automated. If the action is automated, the implementation differs depending upon the operating system:

- UNIX: daemons are usually controlled by the `inetd` daemon. This daemon controls and supervises the system and communication daemons.
- Windows NT: such programs are services (as of version 1.00.xx, the server is not a system service).

2. A browser issues a request to the Web server.

The Web server (also called `httpd`) communicates via TCP/IP. The default port is usually 80.

3. The Web server detects whether the request has been made to a 4GL program or a static HTML page.

If the request is made to a 4GL application, the Web server will create an instance of the HTML client (**fglcl**).

4. The HTML client can now contact and fetch information from the Web server.

The behavior of the HTML client is defined according to the HTML client's configuration file and the argument given by the Web server.

Communication between the client and the server is through sockets.

5. If this is a new request, the HTML server creates a new instance of **fglrun** (the application server process).

If this is a continuation of a previous process, it sends the data to the application server handling this browser client. In either case, a new Web page is created from the output of the application server.

6. The HTML server sends a message to the new application server process and receives a reply.

7. The HTML server processes the message from the application server process.
Communication between the server, the HTML server (**fglhtml**), and the application server process (**fglrun**) is done with anonymous pipes.
8. The HTML server sends the Web page to the client.
9. The HTML client sends the Web page to the Web server.
10. The page is then sent to the browser in the same way as a normal HTML page.
Communication between the Web server and the browser uses the HTTP protocol.
11. The application server process receives information from the HTML server (**fglhtml**) that terminates it (the program ends, or the user exits the program).
12. The server receives termination information from the application server process and sends the normal termination page to the client.
13. The client sends this page to the Web server, which sends it to the browser in the same way as a normal HTML page.
14. The session ends.

Enhancing the Application Interface

The Web deployment software provides mechanisms to enhance the appearance of your applications for display in a Web browser.

You can improve the appearance of your application in the following ways:

- Use your own page frames in the header and footer.
- Enhance your screens by inserting HTML tags in your **.per** files.
- Modify the application itself to improve its appearance.

Supplying Your Own Headers and Footers

Standard page headers and footers are generated by default; however, you can supply your own headers and footers. The Dynamic 4GL HTML server will load the header and footer from the configuration file.

For example, you can add a corporate-style page frame with logo, background, title, standard references to the home-page and other information.

HTML Tags

You can use standard HTML tags in your applications. In addition, you can represent tables using HTML emulation.

Disabling Password Display

To design your interface, you can use the new tag “Password”. Use the class=“Password” parameter in the .per file to disable the display of passwords in your application. The parameter generates the HTML tag <INPUT type=password ... >.

Using the “Password” tag, you can disable the display of the user’s password (the password is replaced by asterisk characters). Include the following code in the .per file:

```
f005 = customer.password,class="Password",invisible,not  
null,required;
```

The following tag is generated:

```
<INPUT TYPE="password" NAME="Ef005_11" SIZE=15 VALUE="">
```

Similarities Between a .per File and an .html File

A .per file is a screen description file for 4GL. An .html file is a screen and resource description file for any Web browser. Both file types are in ASCII format.

The following table summarizes the differences between the two types of files.

.per File	.html File
User interface is done through screen.	User interface is done through pages.
A field can be either dependent on a database or completely independent.	A field is independent of any data source.
A form might or might not depend on a table.	A page does not depend on any table.
Forms are defined by non-proportional ASCII characters.	Pages are defined by proportional characters and special HTML tags.
International characters are quite difficult to manage.	All international characters are a sequence of ASCII characters. For example, "é" is represented by "é".
Forms are not expandable without modification of the form compiler.	HTML is <i>expandable</i> through new tags that can be interpreted by specific software (Web form compiler, browser, and so forth).
There is a notion of forms in HTML.	An HTML form is contained in a page. A page can contain more than one HTML form. An HTML form contains several fields, buttons, and widgets.

Deploying a Sample Application

This section uses a simple Internet phonebook application to illustrate the steps required to deploy your Dynamic 4GL application on the Web. This section assumes the following facts:

- Dynamic 4GL is installed.
- You understand the operation of **fglcl** and **fglhtmlcl**.
- You know the location of the configuration files.
- You have a basic knowledge of HTML.

The example covers the following steps:

1. Creating your Dynamic 4GL application.
2. Editing your server configuration file.
3. Creating a script to initialize the application.
4. Editing your client configuration file.
5. Starting the HTML server daemon.
6. Starting the browser.
7. Using the application from within the browser.
8. Enhancing the application to optimize it for Web use.

The application contains the following modules (**.4gl**):

- **browse**: Handles browsing in companies and contacts. It uses simple DISPLAY ARRAYS.
- **formgen**: Creates the forms shown in the application.
- **globals**: Contains the variables that must be global to a project or set of projects.
- **init**: Where initialization takes place. In this application, it is used for key (button) mapping.
- **main**: Handles simple initialization and menu generation.
- **new**: Creates new companies and contacts.
- **show**: Displays the complete information list of companies and contacts. It is also used to edit and delete companies and contacts.
- **tools**: A library module that contains SuperUser(), a function that checks the super user's login name and password. In this version, only a basic authentication scheme is used. Login and password are hard coded in the source code (and thus cannot be changed easily).

Screens

A screen file has the extension **.per**. The following screens are used in the application:

- **fcompany**: Includes company information.
- **fcontact**: Includes contact information.
- **flcomp**: Displays a list of companies through a screen record that contains the company's unique id, name, telephone number, fax number, and email address.
- **flctct**: Lists some entries of the contact such as unique id, name, telephone number, fax number, and email address.

The contact's name is the result of a concatenation of a title (Mr., Ms., and so forth) and the first, middle, last names, and a suffix.

The screen also has a field for selecting and displaying the company's name.

- **fpasswd**: Fetches the user name and password.

Step 1: Creating a Dynamic 4GL Application

The first step is to create a 4GL application and then recompile it with Dynamic 4GL. For information on creating a 4GL application, refer to your 4GL documentation.

The following 4GL application has been created for you. This application helps you to manage company and sales contact information. This application allows users to:

- browse companies.
- browse contacts.
- edit and add companies.
- edit and add contacts.
- remove companies and contacts with privileged access.

Step 2: Editing the Server Configuration File

The UNIX **phonebook.conf** file contains the entries for the sample application:

```
appName="phonebook"  
client="fgld.exe"  
defaultProgram="./start"  
service="6500"  
serverNumber=96  
emulateHTML=1
```

- **appName**: Creates the link between each page.
- **client**: The name of the client that you are using. In this case, combining **appName** with **client** will give you this call: **fgld?phonebook**.
- **defaultProgram**: The script used to initialize the program. This start script is described in step 3.
- **service**: The base port number.
- **serverNumber**: The offset from the base port (6500) used to create the final port number. For example, this file specifies port 6596.
- **emulateHTML**: Instructs the server to emulate HTML automatically.

Examples of Configuration Settings

This section describes updates that you can make to the **appname.conf** file.

Step 2: Editing the Server Configuration File

The following example shows the General Features section of the configuration file for the phonebook example. The file is named **phonebook.conf**.

```
# General features
#####

# Version of the configuration script
version="0.94.2a"

# Application Name
appName="phonebook"

# Script name in the /cgi-bin/ directory
# fglcl for Unix
# fglcl.exe for Unix / Windows NT
client="fglcl.exe"

# Service-name to register the daemon
service="6500"

# The offset from server-name-port
serverNumber=96

# Security Level
securityLevel=1

# Expirations-Time for Application in seconds
timeOut=1200

# Maximum tasks (default : -1)
maxTasks=10

# Debug level 0-none 1-verbose 2-no demonize (foreground)
debug=0
```

Styles Configuration

To change the appearance of the application in the browser, you can edit the Configuration of Styles section in the **appname.conf** file. For example, the **phonebook.conf** file has these values:

```
# Configuration of Styles
#####

# Buttons below form
buttonDown=0

# Errors below form
errorDown=0

# Answer as Multipart/Mime (Use this when uploading of files
needed!)
multipart=0

# Show menu entries as links (not Buttons)
menuAsLink=0

# Width of form's buttons (0 means minimum)
buttonWidth=10

# Width of menu fields (0 means minimum)
menuWidth=0

# HTML Emulation (default : 0)
emulateHTML=0

# Images path (default : "/images")
imagePath="/Cli-HTML/clipart"

# Show alternate text for images (default : 0)
showImageAlternate=1

# Border width of an image when image is a link (default : 2)
imageBorder=2
```

The **emulateHTML** variable must be set to 0 if you want to include HTML tags in your **.per** screen files.

Spawning Method

The spawning method determines how the application is started. The different methods do not interfere with the look and feel of the interface.

```
# Spawning methods
#####

# Spawn method
# 0 : spawned by shell
# 1 : spawned by runner and environment variables
spawnMethod=0

# Script to start the application
defaultProgram="start"

# Runner name
fglrunName=""

# Start module
fglrunTarget=""

# Environment
# Note : do not use environment variables within definition of
environment
fglrunEnv0=""
```

Arrays

You can specify using custom arrays to improve the look and feel of your application in the Arrays section:

```
# Arrays
#####

# Array is seen as a button (default : 0)
arrayAsButton=1

# Image array (default : "/images/bullet.gif")
arrayImage="/Cli-HTML/clipart/phonebook-bullet.gif"
```

Predefined Macros

The Pre- and Post-Page Macros section specifies predefined macros to set:

- Background color for the entire application
- The title
- The string that lets the user run the application again (such as *Try again*)
- Images that appear in the application

The following section from the configuration file **phonebook.conf** shows the Pre- and Post-Page Macros section:

```
# Pre and post page macros
#####
$NEEDED1="Pragma: no-cache
Content-type: text/html
"
$NEEDED2="
<META HTTP-EQUIV=\"Pragma\" CONTENT=\"no-cache\">
<META HTTP-EQUIV=\"Cache-Control\" CONTENT=\"no-cache\">
"
$BACKCOLOR="BGCOLOR=\"#6F6FFF\" "
$title="<TITLE>The Phonebook - Demonstration</TITLE>"
$TRYAGAIN="<A HREF=\"/cgi-bin/fglcl?phonebook\">Try again</A>"
$REFRESH="
<META HTTP-EQUIV=\"REFRESH\" CONTENT=\"10; URL=/cgi-
bin/fglcl?phonebook\">"
$HEAD="<IMG SRC=\"/Cli-HTML/clipart/phonebook-large.gif\"
ALIGN=LEFT>
<H3>The Phonebook</H3>"
$TAIL="<HR>
<CENTER>Welcome to <B>The Phonebook</B> Demo program!</CENTER>
</BODY>
</HTML>"

# Header
headRecord="$NEEDED1
<HTML>
<HEAD>
$NEEDED2
$title
</HEAD>
<BODY $BACKCOLOR>
$HEAD
"

# Tail
tailRecord="
$TAIL"
# Error
```

Step 2: Editing the Server Configuration File

```
errorRecord=
"Pragma: no-cache
Content-type: text/html
<HTML>
<HEAD>
$NEEDED2
$REFRESH
$title
</HEAD>
<BODY $BACKCOLOR>
$HEAD
<BR>
An error has occurred...<BR>
Error %s<BR>
<BR>
$TRYAGAIN
$TAIL
"
# Time Out
timeOutRecord=
"Pragma: no-cache
Content-type: text/html
<HTML>
<HEAD>
$NEEDED2
$REFRESH
$title
</HEAD>
<BODY $BACKCOLOR>
$HEAD
<BR>
This application has been terminated on timeout ...<BR>
$TRYAGAIN
$TAIL
"
# Normal end
endRecord=
"$NEEDED1
<HTML>
<HEAD>
$NEEDED2
$title
</HEAD>
<BODY $BACKCOLOR>
$HEAD
<BR>
Thanks for trying The Phonebook <BR>
$TRYAGAIN
$TAIL
"
```

Results of Updating the Application Configuration File

The modifications shown in the sample configuration file change the sample application display from the display shown in [Figure 10-3](#) to the display shown in [Figure 10-4](#).

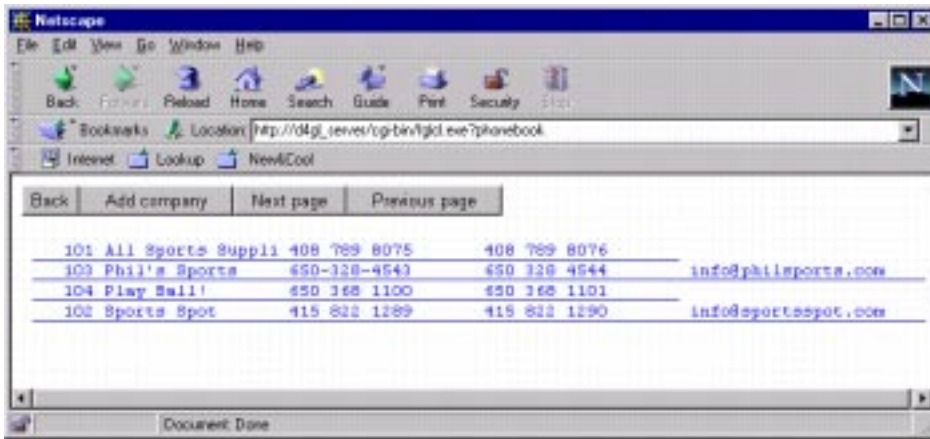


Figure 10-3
Display Before
Modification

Step 2: Editing the Server Configuration File

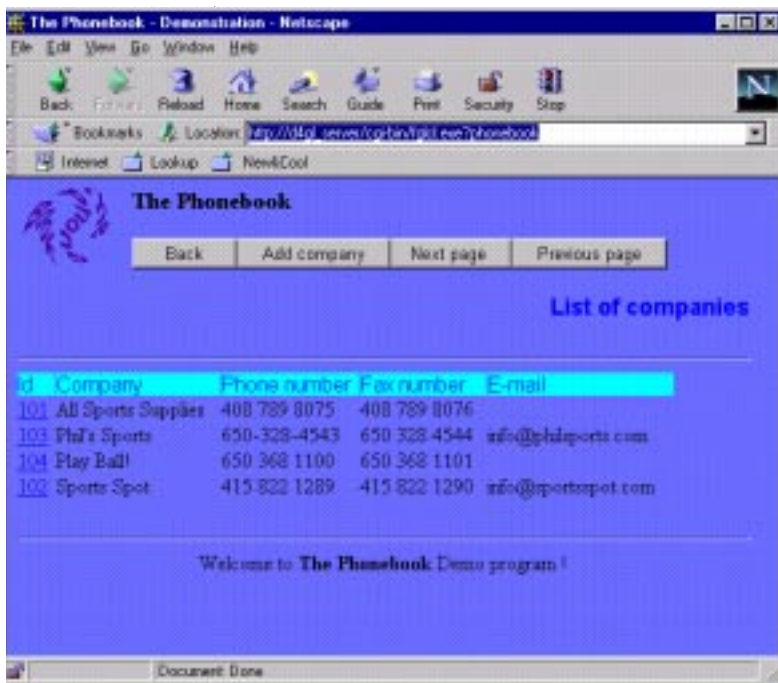


Figure 10-4
Display After
Modification

The following modifications are displayed:

1. A small animated gif
2. A header designed in the **appname.conf** file
3. A small modification in the forms
4. An array header
5. A background color

Step 3: Creating a Script to Initialize the Application

The shell launched to initialize the program on UNIX is named `start` and contains the following code:

```
#!/bin/sh

FGLGUI=2
export FGLGUI

FGLPROFILE=$FGLDIR/etc/fglprofile.web
export FGLPROFILE

exec fglrun main
```

This script sets `FGLGUI` to 2, which is the mode used for HTML applications. It also specifies an `FGLPROFILE` file that is used for HTML applications.

Step 4: Editing Your Client Configuration File

The `fglcl.conf` file is located in the Web server's `cgi-bin` directory. For more information, see the client configuration file, `fglcl.conf`.

Modify the following parameters in the `fglcl.conf` file:

```
phonebook.fglserver=app_server_ip_address:96
phonebook.debug=0
phonebook.HTMLdebug=0
```

Replace `app_server_ip_address` with the IP address of your application server. You can use the name of the application server or a fully qualified hostname and domain name; however, this can require a DNS lookup or a search in `/etc/hosts` on UNIX or in `%WINDIR%\system32\drivers\etc\hosts` on Windows NT, which can slow performance.

Step 5: Starting the HTML Server Process on UNIX

Start the server process on UNIX with the following command:

```
$ fglhtlmd -f phonebook.Unix.conf
```

To display system messages, enter the following command:

```
$ fglhtlmd -d -f phonebook.Unix.conf
```

Step 6: Starting the Browser

To run your application in a Web browser using the HTML client, enter the URL of the Dynamic 4GL application in your browser:

```
http://web_server_ip_address/web_server_cgi_alias/  
fglcl.exe?appname
```

For example:

```
http://d4gl_server/cgi-gin/fglcl.exe?phonebook
```

Step 7: Using the Application

After your browser has started the application, the user can interact with it to perform database operations. In this example, the user can browse through companies as shown in [Figure 10-5](#).

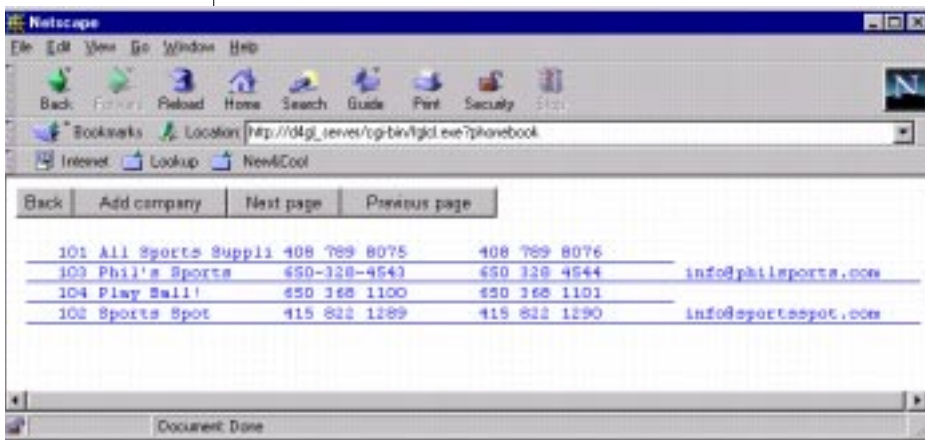
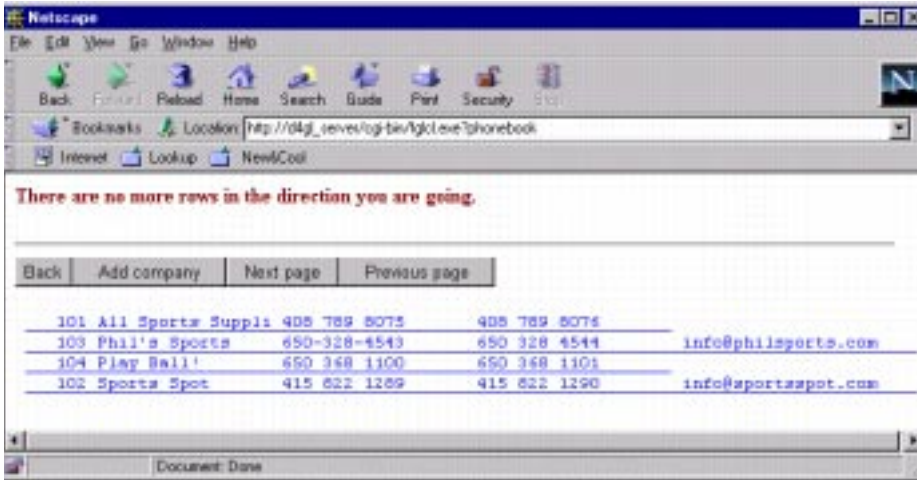


Figure 10-5
Company Browse
List

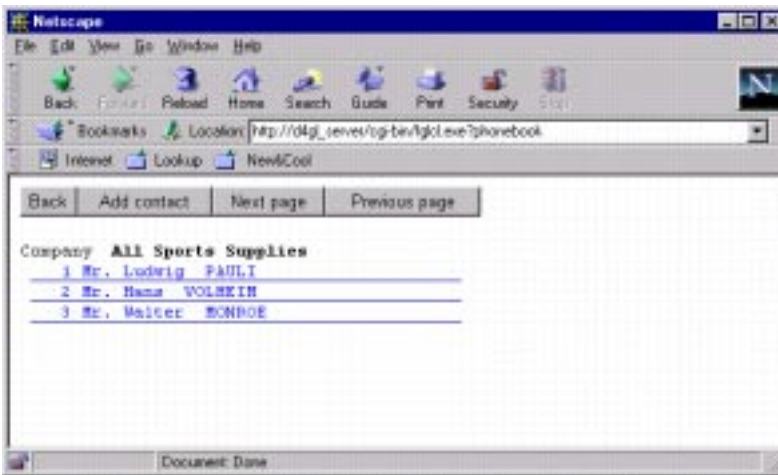
If no more records exists and the user clicks on the Next Page button, the program generates an error, as shown in [Figure 10-6](#).

Figure 10-6
Generated
Error Message



In addition, the user can select a company from the list in the previous display. The user can then display all the company's known contacts, as [Figure 10-7](#) shows.

Figure 10-7
Contact Browse List



Step 7: Using the Application

If you select one contact from the list, the information shown in [Figure 10-8](#) is displayed.

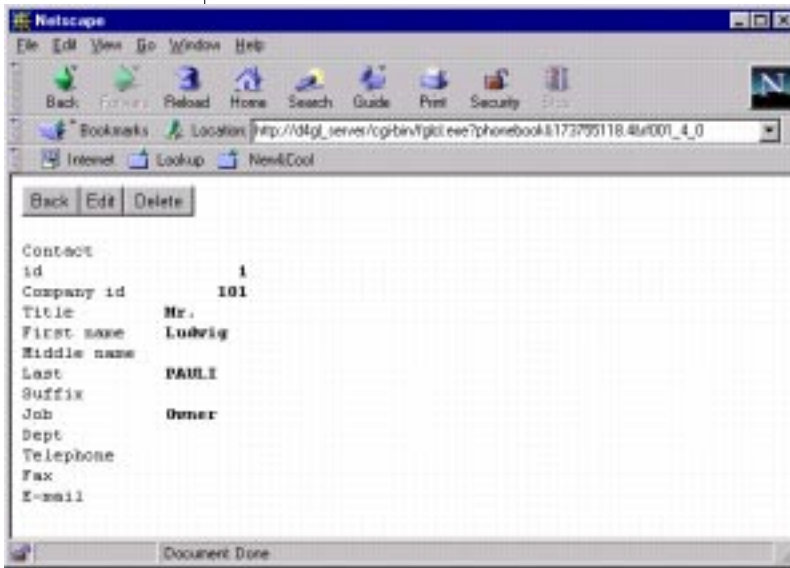


Figure 10-8
Contact Information
Display

To edit the record, click **Edit**. [Figure 10-9](#) appears.

The screenshot shows a Netscape browser window with the address bar displaying `http://101.sever.com/html/tech/homebook`. The main content area contains a form for editing contact information. The form fields are as follows:

Contact id	1
Company id	101
Title	Mr.
First name	Ludwig
Middle name	
Last	PAULI
Suffix	
Job	Owner
Dept	
Telephone	
Fax	
E-mail	LPauli@

Figure 10-9
Contact Information
Record Edit Display

The application automatically generates a list box when the 4GL code uses any `INCLUDE=` statement. List boxes, such as the one in [Figure 10-10](#), are automatically displayed when you use `INCLUDE` in the screen `.per` file.

The screenshot shows a list box with a dropdown menu. The menu items are: Dr., Miss, Mr. (highlighted), Mrs., Ms., and Prof.

Figure 10-10
List Box

This list box was generated from the following code:

```
f003 = formonly.ctc_title, include=(
  "", "Dr.", "Miss", "Mr.", "Mrs.", "Ms.", "Prof.");
```

Step 8: Enhancing the Application

You can enhance your Dynamic 4GL application for deployment on the Web by:

- enhancing the screen files.
- creating email and Web site links.
- using features specific to Dynamic 4GL.
- improving your application interface.

Creating Email and Web Site Links

You can add features such as URLs or email addresses. To do this, the database must contain records for the company's Web site and email address. You can then use the following code to create a link to an address:

```
<A HREF="mailto:support@informix.com">Send a mail to support</A>
```

To link to a new page:

```
<A HREF="http://www.informix.com">See our site !</A>
```

The 4GL code can be enhanced as follows to incorporate links that allow you to send email to the company or jump to the company's home page:

```
DEFINE l_buffer CHAR(500)

...

IF r_company.com_email IS NOT NULL THEN
  LET l_buffer = "<A HREF=\"mailto:\",
    r_company.com_email CLIPPED,
    \">\",r_company.com_email CLIPPED, "</A>"
  DISPLAY l_buffer TO com_email
END IF

IF r_company.com_web IS NOT NULL THEN
  LET l_buffer = "<A HREF=\"http://\",
    r_company.com_web CLIPPED, \"\"
    TARGET=\"_blank\">\", r_company.com_web CLIPPED, "</A>"
  DISPLAY l_buffer TO com_web
END IF
```

To send email, the user can click on the email link. The Web field links to the company's Web page.

Enhancing the Screen Files

Enhancing screen files involves adding HTML tags. You must first set the `emulateHTML` parameter to 0 in the `appname.conf` file in order to specify that the HTML server will read HTML code.

The following example is a basic input/edit/display form without HTML:

```

DATABASE formonly

SCREEN
{
Contact

    id          [f001   ]
    Company id  [f002   ]
    Title       [f003   ]
    First name  [f004           ]
    Middle name [f005           ]
    Last        [f006           ]
    Suffix      [f007   ]
    Job         [f008           ]
    Dept        [f009           ]
    Telephone   [f010           ]
    Fax         [f011           ]
    E-mail      [f012           ]
}
END

ATTRIBUTES
f001 = formonly.ctc_id;
f002 = formonly.ctc_com_id;
f003 = formonly.ctc_title, include=(
    "", "Dr.", "Miss", "Mr.", "Mrs.", "Ms.", "Prof.");
f004 = formonly.ctc_first;
f005 = formonly.ctc_middle;
f006 = formonly.ctc_last;
f007 = formonly.ctc_suffix, include=(
    "", "I", "II", "III", "IV", "Jr.", "Sr.");
f008 = formonly.ctc_job;
f009 = formonly.ctc_dept;
f010 = formonly.ctc_tel;
f011 = formonly.ctc_fax;
f012 = formonly.ctc_email;
END

INSTRUCTIONS
DELIMITERS " "
END

```

The following example is the same form with HTML enhancements:

```

DATABASE formonly

SCREEN
{
<p align="right">
<big><font face="Arial" color="#0000FF">
<strong>Contact</strong></font></big></p>
<HR>
<TABLE>
<TR><TD>id                </TD><TD>[f001      ]</TD></TR>
<TR><TD>Company id        </TD><TD>[f002      ]</TD></TR>
<TR><TD>Title              </TD><TD>[f003      ]</TD></TR>
<TR><TD>First name         </TD><TD>[f004      ]</TD></TR>
<TR><TD>Middle name        </TD><TD>[f005      ]</TD></TR>
<TR><TD>Last                </TD><TD>[f006      ]</TD></TR>
<TR><TD>Suffix              </TD><TD>[f007      ]</TD></TR>
<TR><TD>Job                  </TD><TD>[f008      ]</TD></TR>
<TR><TD>Dept                 </TD><TD>[f009      ]</TD></TR>
<TR><TD>Telephone            </TD><TD>[f010      ]</TD></TR>
<TR><TD>Fax                   </TD><TD>[f011      ]</TD></TR>
<TR><TD>E-mail                </TD><TD>[f012      ]</TD></TR>
</TABLE>
}
END

ATTRIBUTES
f001 = formonly.ctc_id;
f002 = formonly.ctc_com_id;
f003 = formonly.ctc_title, include=(
"", "Dr.", "Miss", "Mr.", "Mrs.", "Ms.", "Prof.");
f004 = formonly.ctc_first;
f005 = formonly.ctc_middle;
f006 = formonly.ctc_last;
f007 = formonly.ctc_suffix, include=(
"", "I", "II", "III", "IV", "Jr.", "Sr.");
f008 = formonly.ctc_job;
f009 = formonly.ctc_dept;
f010 = formonly.ctc_tel;
f011 = formonly.ctc_fax;
f012 = formonly.ctc_email;
END

INSTRUCTIONS
DELIMITERS " "
END

```


The following section describes the HTML tags shown in this example.

Title

The first set of tags defines a right-aligned paragraph using the Arial font (similar to Helvetica) in blue (#0000FF).

```
<p align="right">
<big><font face="Arial" color="#0000FF">
<strong>Contact</strong></font></big></p>
```

The value #0000FF defines a dark blue color.

Horizontal split

The HTML HR tag creates a horizontal line on the page.

```
<HR>
```

Table

The HTML TABLE tag creates a new table.

```
<TABLE>
<TR><TD>id           </TD><TD>[ f001   ]</TD></TR>
<TR><TD>Company id  </TD><TD>[ f002   ]</TD></TR>
<TR><TD>Title       </TD><TD>[ f003   ]</TD></TR>
<TR><TD>First name  </TD><TD>[ f004   ]</TD></TR>
<TR><TD>Middle name </TD><TD>[ f005   ]</TD></TR>
<TR><TD>Last        </TD><TD>[ f006   ]</TD></TR>
<TR><TD>Suffix      </TD><TD>[ f007   ]</TD></TR>
<TR><TD>Job         </TD><TD>[ f008   ]</TD></TR>
<TR><TD>Dept        </TD><TD>[ f009   ]</TD></TR>
<TR><TD>Telephone   </TD><TD>[ f010   ]</TD></TR>
<TR><TD>Fax         </TD><TD>[ f011   ]</TD></TR>
<TR><TD>E-mail      </TD><TD>[ f012   ]</TD></TR>
</TABLE>
```

Tables allow good positioning while still using proportional fonts.

The display is shown in [Figure 10-11](#).

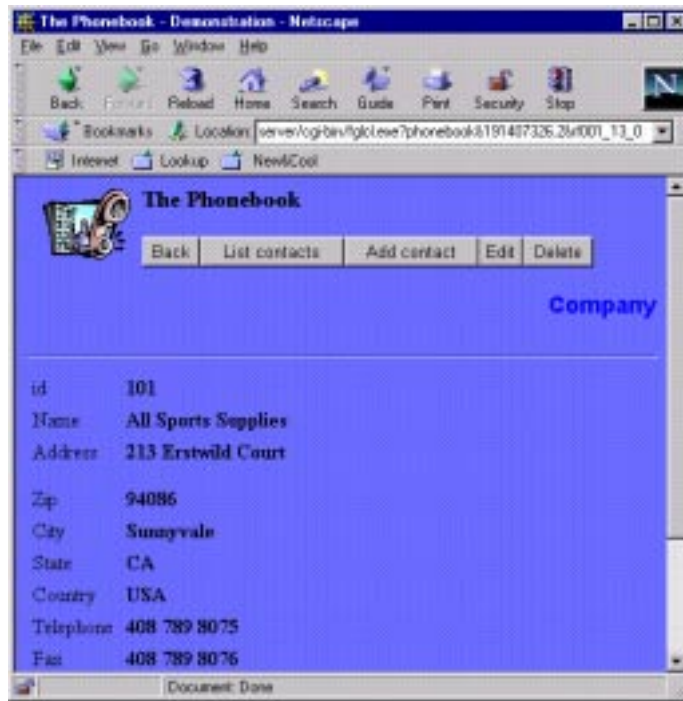


Figure 10-11
Form Using an
HTML Table

How Links Between Pages Work

The HTML client stops after each transaction. A link, however, must be made as a follow-up.

Links between pages are based on the process id (enciphered) of the application server process (**fglrun**) and a sequence number.

Nothing appears on the client side, so you cannot copy and paste the URL to another browser. The only visible item is the application name, which has the form: **fglcl?appname** or **fglcl.exe?appname**.

HTML Emulation for Tables

You can insert HTML tags for tables in your **.per** file in order to design a more attractive field display. The HTML client will automatically add the necessary HTML tags that your application requires. This is called HTML emulation.

For example, if you include the following HTML tags in your **.per** file:

```
<TABLE WIDTH=70% BORDER CELLPADDING=2 CELLSPACING=0
BGCOLOR="#FFFFFF" >
<TR BGCOLOR="#D0D0D0" >
<TH WIDTH=30%>&nbsp;&nbsp; <TH>&nbsp;&nbsp; 
Name [ f001 ]
Password [ f005 ]
</TABLE>
```

Your application displays the table shown in [Figure 10-12](#).

Customer Login	
Name	<input type="text" value="admin"/>
Password	<input type="password" value="*****"/>

Figure 10-12
HTML Table Display
Using HTML
Emulation

Dynamic 4GL Features

The following sample code generates a new screen from the data in the company table and creates a new file. The function **generateForms()** is added to the **.per** file.


```

CALL channel::write ("f1", "</TR> </TABLE>")
CALL channel::write ("f1", "}")
CALL channel::write ("f1", "END")
CALL channel::write ("f1", "")
CALL channel::write ("f1", "ATTRIBUTES")
CALL channel::write ("f1", "c001 = formonly.company_name,
include=("")

# Include section from table
PREPARE sqlStatement FROM "SELECT com_name FROM company ORDER BY
com_name"
DECLARE sqlCursor CURSOR FOR sqlStatement
OPEN sqlCursor
FETCH sqlCursor INTO l_buffer
WHILE status <> NOTFOUND
    LET l_writeBuffer = "\", l_buffer CLIPPED, "\", "
    CALL channel::write ("f1", l_writeBuffer CLIPPED)
    FETCH sqlCursor INTO l_buffer
END WHILE

FREE sqlStatement
FREE sqlCursor

# Tail
CALL channel::write ("f1", "\"\"")
CALL channel::write ("f1", ");")
CALL channel::write ("f1", "f001 = formonly.contact_id;")
CALL channel::write ("f1", "f002 = formonly.contact_name;")
CALL channel::write ("f1", "f003 = formonly.contact_tel;")
CALL channel::write ("f1", "f004 = formonly.contact_fax;")
CALL channel::write ("f1", "f005 = formonly.contact_email;")
CALL channel::write ("f1", "END")
CALL channel::write ("f1", "")
CALL channel::write ("f1", "INSTRUCTIONS")
CALL channel::write ("f1", "DELIMITERS \" \")
CALL channel::write ("f1", "SCREEN RECORD scr[10] (")
CALL channel::write ("f1", " formonly.contact_id,")
CALL channel::write ("f1", " formonly.contact_name,")
CALL channel::write ("f1", " formonly.contact_tel,")
CALL channel::write ("f1", " formonly.contact_fax,")
CALL channel::write ("f1", " formonly.contact_email)")
CALL channel::write ("f1", "END")
CALL channel::write ("f1", "")
CALL channel::CLOSE ("f1")

RUN "fglform frm1listcontact.per" RETURNING i

END FUNCTION

```

When the **generateForms()** function is called, this source generates a new form using channels and a call to the form compiler, **fglform**. The result of the modification is shown in [Figure 10-13](#).

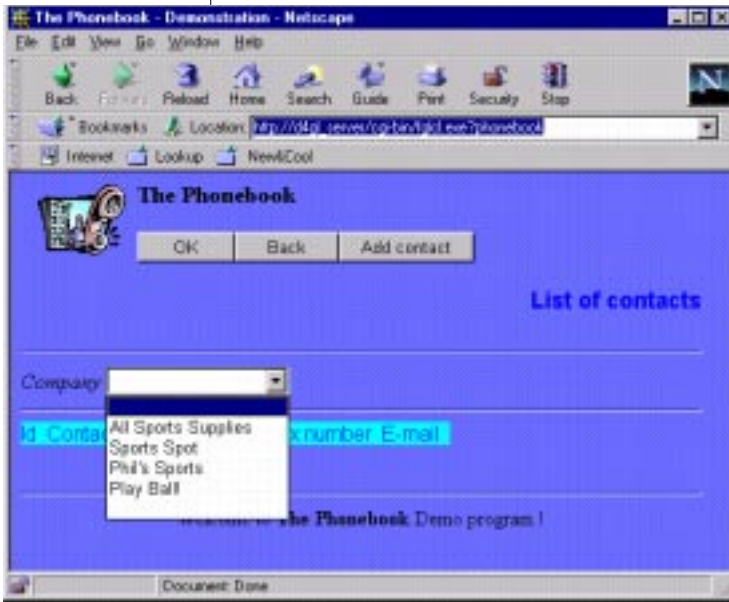


Figure 10-13
New Form
Generated by
`generateForms()`
Function

Security Levels

This section describes the levels of security features that you can implement.

Default Security

The HTML server identifies the client by encoding in the Web page a special key that allows tracing. The key is encoded to prevent it from being duplicated by another browser, thus preventing a second connection to the application server.

Figure 10-14 shows the default security architecture of the Web deployment software when a Dynamic 4GL application is running on the Web.

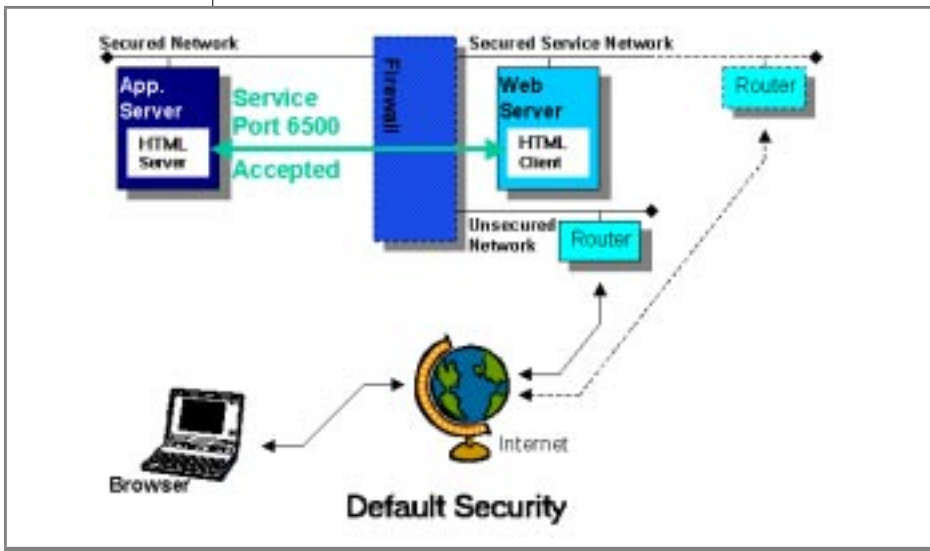


Figure 10-14
Default Security
Architecture for
Web Deployment

Notice that:

- the application server is on a secured network with access to the Internet.
- the Web server is on a secured service network.
- the router can either be on a secured service network or on a third, unsecured network.
- the firewall is optional.
- your application server and your Web server can reside on the same computer.

Each runner process started by the application server has a unique, random number. This number is used to make a link between each page the application server serves to the Web server.

Recommendations for Enhancing Security

The following recommendations can enhance the security of your applications.

SSL

Using a secure socket layer (SSL) between the Internet browser and the Web server facilitates a secure data flow.

Using a Filtering Router

A filtering router can disable port 6500 (the standard application service port) or the effective application server communications port on the router. After this port is disabled, it prevents access to the application server.

With the application service port disabled, normal transactions on port 80 (the standard HTTP service port) are still allowed. This allows the user's browser to have access to Web server documents.

Using a Firewall

A firewall can restrict communication so that only the Web server can communicate with the application server. Any requests from other hosts to the application server are stopped by the firewall.

Application, Web Server, and Database Security

This section describes security features and considerations for Web Deployment. The following features help ensure the security of your database applications:

- No database network access (such as SQL-Net/ODBC) is necessary.
- No direct Internet connection is needed for the application or database server.
- Only a small amount of code (the HTML client process) exists in the CGI binaries directory on the Web server.
- Only one trusted channel is used to traverse the firewall from the Web server (proved) to the HTML server (also proved).

- Logging of the Dynamic 4GL interactions is possible.
- Applications can run in a special, definable environment with special and limited rights.
- Runtime system messages, alerts, and errors are not processed by the HTML server and thus are not forwarded to the client or visible through your browser. Isolating the messages has the advantage of leaving your system anonymous.

Certificate Authority

In order to use SSL, you need to ask a Certificate Authority to sign your X509 v3 certificate.

For more information on encryption support and restrictions, see Netscape's Export Restrictions on International Sales at the following URL:

<http://developer.netscape.com/docs/manuals/security/exprt/index.htm>

Preventing Security Problems

The following list summarizes some methods that can be used to prevent security problems:

- Reading the data flow between browser and Web server
SSL will prevent eavesdropping of data.
- Unauthorized entry into the application server code
 - Router filtering will ignore all TCP/IP packets to port 6500 of the application server.
 - The firewall will ignore all TCP/IP packets to port 6500 of the application server coming from any host other than the Web server.
- Copying the URL to another browser
The basic Dynamic 4GL HTML server mechanism will reject the copied URL.

- Reading the data flow between the HTML client and the HTML server. Anyone attempting to break into the system must gain control of a computer based on the secured network or on the secured service network.
- Denial of service
The HTML Client is a small, connectionless program that can only transmit authorized packets to the HTML server. Therefore, even if the client stops functioning, the server will still be accessible.

Configuring the Web Deployment Software

This section describes the configuration settings for Web deployment. The following files contain these settings:

- **appname.conf** file
- **fglcl.conf** file: use the **fglcl.conf** file to configure the HTML Client.

If you plan to have a great number of users processing your program, you can have them use more than one **fglhtml** daemon, and consider one program as more than one application. Specify this in the **fglcl.conf** file.

Configuration Settings in the **fglcl.conf** file

To configure the HTML Client, use the **fglcl.conf** file.

Location

A multiple-entry file, such as an **.ini** file, would allow the HTML client (**fglcl**) to find its required information.

For example:

```
sample.fglserver=198.2.1.0:0  
computer.fglserver=198.2.1.0:1
```

Calls would then be done through **fglcl?sample** or **fglcl?computer**.

The full URL could then look like the following:

```
http://web_server_ip_address/web_server_cgi_alias/fglcl?appName
```

On Windows NT, the **fglcl.conf** file is not in the registry because spawned cgi binaries might not have access to the Registry. Also, an unknown user could by means of a program read the registry.

The best portable solution is to place the file HTML client (**fglcl**) program in the CGI binaries directory for the following reasons.

Some Web servers have a WWW_ROOT like the FTP_ROOT for FTP servers and spawned processes cannot access files and directories that are at a higher directory level than WWW_ROOT.

Accessing this file through a Web browser will make the system execute rather than read this file (all files in the CGI directory are considered executable).

The full syntax for **fglcl.conf** is:

```
appName.fglserver={app_server_ip_address}:{port - 6500}
[appName.debug={0|debug_level}]
[appName.HTMLdebug={0|1}]
```

fglserver

This variable uses the same configuration as the FGLSERVER variable. It must be written **fglserver** (lowercase). The port number must be specified and is always -6500.

For example, if you plan to run on your computer **app_server** whose IP address is 198.100.150.4, on port 6542, the variable entry would be:

```
fglserver=198.100.150.4:42
```

debug

This variable indicates debug level. Those debug traces are seen on the standard error (stderr) and will not be seen in the HTML page.

Debugging can degrade performance.

HTMLdebug

This flag should be set to 0 (no debugging) or to 1 (debugging).

A setting of 1 will show some debug traces of the client (fglcl) in the browser.

Security

This section contains some notes about security features in various Web servers.

Apache Web Server

Trying to access the **fglcl.conf** file through the Web server will generate the following error:

```
Forbidden
You do not have permission to access /cgi-bin/fglcl.conf on this
server.
```

Microsoft IIS/Personal Web Server 4.0

When you use the normal configuration with the **fglcl.conf** file in the CGI binaries directory on IIS/PWS 4.0, the browser displays the following prompt:

```
You have chosen to download a file from this location

fglcl.conf from axis

What would you like to do with this file?
```

You have two options:

- Open this file from its current location
- Save this file to disk

The following two considerations regarding **fglcl.conf** and security on Windows NT should be observed:

- The **fglcl.conf** file is an HTML file and its access should be restricted through the Web server.
- The **fglcl.conf** file is a system file and it should be protected (or secured) as a normal system file.

Security Through the Web Server

You can set security for the **fglcl.conf** file by launching Microsoft Management Console. Find the **fglcl.conf** file in the file list and look at the file's properties. You will see the dialog box shown in [Figure 10-15](#).

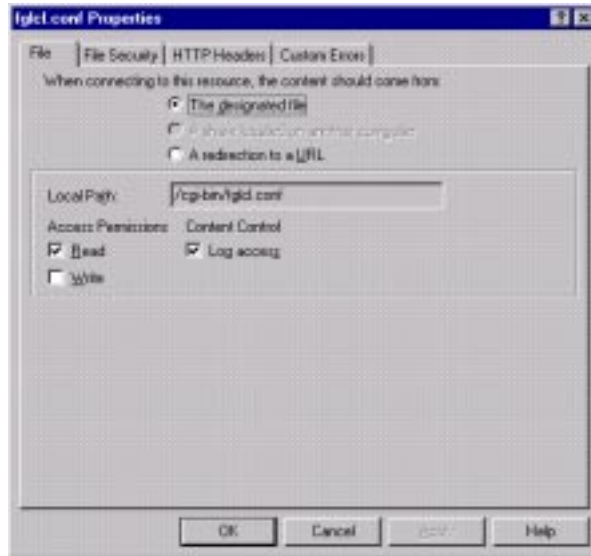


Figure 10-15
*fglcl.conf Properties
Dialog Box*

On the **File** tab, uncheck Read Access Permissions checkbox.

You can also go the **File Security** tab and click **Edit** to change the "Anonymous Access and Authentication Control" configuration.

In the Authentication Methods dialog box that appears, you can disable (uncheck) all the options.

Security Through the File System

Make sure that the spawned process (**fglcl.exe**) can read the **fglcl.conf** file with the rights from the standard internet guest user. Spawned processes from the Web server will be under this user's rights. To specify file permissions, right-click on the **fglcl.conf** file. Select Properties and then select the **Security** tab. Click **Permissions** to display the File Permissions dialog box.

Summary

When everything is set properly, you will be prompted for a password when trying to access this file from another station.

Internet Explorer and IIS/Personal Web Server have special features that allow them to send and receive data more securely and thus will let the current user on the Web server access (read) the **fglcl.conf** file without any problems. These special features are not available if you are using Navigator with the IS server.

Configuring the appname.conf File

The naming convention for the application configuration file is **appname.conf**, where appname is the name of the application. The configuration variables are not case sensitive.

The **appname.conf** file has the following flags:

- General configuration settings
- Pre and post messages
- Styles
- Spawning
- Arrays

General Configuration Settings

You can set the following general configuration flags:

- version: Version
- appName: Application name
- client: Client
- service: Service
- serverNumber: Server number
- securityLevel: Security level
- timeOut: Time out

- `maxTasks`: Maximum tasks
- `debug`: Debug

Version

Allows you to specify the version number of the configuration file.

For example:

```
# Version of the configuration script
version="1.05"
```

Expected type: string.

Application Name

Allows you to establish a link between the name of the HTML client (**`fglcl`**, **`fglcl.exe`**) and the name of the application.

For more information, see [“How Links Between Pages Work” on page 10-40](#).

For example:

```
# Application Name
appName="phonebook"
```

Expected type: string.

Client

Allows you to establish a link between the name of the HTML client (**`fglcl`**, **`fglcl.exe`**) and the name of the application.

For more information, see [“How Links Between Pages Work” on page 10-40](#).

For example:

```
# Program name in the /cgi-bin/ directory
client="fglcl.exe"
```

Expected type: string.

If you are using the HTML client in a mixed environment (that is, your Web server is on both UNIX and Windows NT), remember that Windows NT executables have an **`.exe`** extension.

You will need to rename your UNIX HTML client to match the name it has under Windows NT.

The HTML client name does not need to be **fglcl.exe**. Any filename can be specified, but the name needs to be the same in all configuration files.

Service

The port number that the daemon uses establishes a connection with the client. The default value is 6500.

For example:

```
# Service name to register the daemon
service="6500"
```

Expected type: integer.

Server Number

The default base address is 6500. You can set it to a different port by specifying a value from 0 to 99, which is added to the base 6500 value.

For example:

```
# The offset from server name port
serverNumber=0
```

Expected type: integer.

Security Level

Allows you to define security levels. The two possible settings are 1 for Level 1 (basic) and 2 for Level 2 (advanced).

For example:

```
# Security Level
securityLevel=1
```

For a full explanation, refer to [“Security Levels” on page 10-44](#).

Important: *The value of securityLevel defaults to level 1.*

Expected type: integer.



Time Out

Allows you to specify the expiration time (in seconds) of the program when the program is inactive. By default, **fglhtml** stops the program if it is inactive for 300 seconds.

For example:

```
# Expiration time for application, in seconds
timeOut=300
```

Expected type: integer.

Maximum Tasks

Allows you to specify the maximum number of tasks that the HTML server (**fglhtml**) can handle.

This setting is used to limit the number of users for a Web application so that it does not interfere with other applications. If you do not want to set a limit, set this value of `maxTasks` to `-1`.

For example:

```
# Maximum tasks (default : -1)
maxTasks=3
```

Expected type: integer.

Debug

Allows you to set the debug level. The default value is 2.

For example:

```
# Debug level
# 0 none
# 1 verbose
# >= 2 no daemon (runs in foreground)
debug=2
```

Expected type: integer.

Pre and Post Messages

The following flags allow you to specify the standard messages your application will display, depending on events raised while the program runs:

- headerRecord: Header
- tailRecord: Tail (footer)
- errorRecord: Error
- timeoutRecord: Time-out message
- tooManyRecord: Too many tasks
- endRecord: Normal termination

Header

Allows you to specify a header. The header is text displayed at the top of the page. It is displayed when a page is generated.

Use the \ character before a double quote (\").

For example:

```
# Header
headRecord=" $NEEDED1
<HTML>
<HEAD>
$NEEDED2
$TITLE
</HEAD>
<BODY $BACKCOLOR>
$HEAD
```

Expected type: message.

Footer

Allows you to specify a footer (or “tail record”). The footer can include a signature, a company logo, or other information that you want to include at the bottom of each HTML page.

For example:

```
# Tail
tailRecord="
<HR>
</BODY>
</HTML>
"
```

Expected type: message

Error

Allows you to add a % character in the text message. This allows you to see the error message generated by the program.

For example:

```
# Error
errorRecord=
"Pragma: no-cache
Content-type: text/html
<HTML>
<HEAD>
$NEEDED2
$REFRESH
$title
</HEAD>
<BODY $BACKCOLOR>
$HEAD
<BR>
An error has occurred...<BR>
Error %s<BR>
<BR>
$TRYAGAIN
$TAIL"
```

Expected type: message.

Time-Out Message

Allows you to specify a time-out message. After a time out (specified by the **timeOut** variable), users will receive this error message if they try to continue to access pages.

For example:

```
# Time-Out Message
timeOutRecord=
"Pragma: no-cache
Content-type: text/html
<HTML>
<HEAD>
$NEEDED2
$REFRESH
$TITLE
</HEAD>
<BODY $BACKCOLOR>
$HEAD
<BR>
This application has been terminated on timeout ...<BR>
$TRYAGAIN
$TAIL
"
```

Expected type: message.

Too Many Tasks

Allows you to specify an error message that is displayed when the maximum task limit is reached, as specified by the **maxTasks** variable.

For example:

```
# Too many tasks message
tooManyRecord=
"Pragma: no-cache
Content-type: text/html

<HTML>
<HEAD>
<TITLE>Too Many Tasks</TITLE>
</HEAD>
<BODY>
<H1>The maximum task limit has been reached.</H1>
</BODY>
</HTML>"
```

Expected type: message.

Normal Termination

Allows you to specify a termination message. After a normal ending of the program, you receive this termination message.

For example:

```
# Normal end
endRecord=
"$NEEDED1
<HTML>
<HEAD>
$NEEDED2
$TITLE
</HEAD>
<BODY $BACKCOLOR>
$HEAD
<BR>
Thanks for trying The Phonebook <BR>
$TRYAGAIN
$TAIL"
```

Expected type: message.

Styles

The following styles are used to configure your application by changing its look and feel:

- **buttonDown**: Button down
- **errorDown**: Error down
- **menuAsLink**: Menu as link
- **buttonWidth**: Button width
- **menuWidth**: Menu button width
- **emulateHTML**: Emulate HTML
- **imagePath**: Image path
- **showImageAlternate**: Image alternate text
- **imageBorder**: Image border

Button Down

Allows you to place buttons on top of or below the form. Specify 0 for buttons below the form and 1 for buttons above the form.

For example:

```
# Buttons below form  
buttonDown=0
```

Expected type: integer.

Error Down

Allows you to set the error line at the top of your page or at the bottom. At the bottom it looks like a standard 4GL program, but it can be difficult to read in a Web browser. Specify 0 to set the error line at the bottom of the page, and 1 to set it at the top of the page.

For example:

```
# Errors below form  
errorDown=0
```

Expected type: integer.

Menu as Link

Allows you to use the menus either as a set of buttons or as text links.

For example:

```
# Show menus as links (not Buttons)  
menuAsLink=1
```

Expected type: integer.

Button Width

Allows you to specify the width, in characters, of on-screen buttons.

```
# Width of buttons on a form (0 means minimum)  
buttonWidth=10
```

Expected type: integer.

Menu Button Width

Allows you to specify the width, in characters, of menu buttons.

For example:

```
# Width of fields menus (0 means minimum)
menuWidth=0
```

Expected type: integer.

Emulate HTML

Allows you to enable HTML emulation.

When you deploy an existing 4GL application on the Web, this application looks like a flat ASCII application.

You can insert HTML tags in order to enhance the look and feel of your application. If you do so, the internal mechanism for emulating HTML must be disabled (set emulateHTML to 0). A value of 1 indicates that the HTML client will emulate (when possible) HTML tags that might not be compatible with the ones that you inserted.

The default value is 0.

For example:

```
# Emulation of HTML
emulateHTML=1
```

Expected type: integer.

Image Path

Allows you to specify the default path where images are located. This setting is used with the Dynamic 4GL DISPLAY statement.

For example:

```
# Images path (default : "/images")
imagePath="/Cli-HTML/images"
```

If you specify the following:

```
imagePath="/Cli-HTML/images"
```

and your Dynamic 4GL program contains the following code:

```
DISPLAY "mypicture.gif" TO myField
```

Then the resulting HTML code will appear as follows:

```
<IMG SRC="/Cli-HTML/images/mypicture.gif">
```

The Web server will then look for the image in <document_root>/Cli-HTML/images/mypicture.gif.

Expected type: string.

Important: Be sure to select an image format that your Web browser supports.



Image Alternate Text

Allows you to specify whether alternate text is displayed. A browser can display alternate text when the image fails to load properly. It can also display alternate text when the cursor is positioned over the image. This text is described in the ALT section of the tag. Specify 1 to display alternate text, or 0 if you do not want alternate text to be displayed.

For example:

```
# Show alternate text for images (default : 0)  
showImageAlternate=1
```

Expected type: integer.

Image Border

Allows you to specify the width of the border, in pixels, that browsers display around images.

For example:

```
# Border width of an image when image is a link (default : 2)  
imageBorder=2
```

Expected type: integer.

Spawning

Spawning happens when the HTML server (**fglhtml**d) starts a new task:

- **spawnMethod**: Spawning method
- **defaultProgram**: Program
- **fglrunName**: Runner name
- **fglrunTarget**: Runner target
- **fglrunEnv**: Runner environment

Spawning Method

The Web deployment software on UNIX systems can use either of two spawning methods for launching a sub-process:

- Spawn using a shell where you must define your environment variables.
- Spawn using a direct call to the runner and defining the environment before calling the runner.

```
# Spawn method (default : 0)
# 0 : spawned by shell
# 1 : spawned by runner and environment variables
spawnMethod=0
```

Expected type: integer.

On Windows NT systems, the spawning method must be set to 1.

Program

Allows you to specify the script with which to start your application.

If you spawn your runner using method 0 (spawn using a shell), the daemon first launches a starting shell that contains **fglrun** and the environment for the whole session. You need to use the **defaultProgram** variable to specify the starting script.

For example:

```
# Script to start the application
defaultProgram="./start"
The start script looks like the following:
#!/bin/sh
FGLGUI=2
export FGLGUI
FGLPROFILE=$FGLDIR/etc/fglprofile.web
export FGLPROFILE
FGLLANG=english
export FGLLANG
unset DBPATH
exec fgldrun main
```

Expected type: string.

Runner Name

Allows you to specify a runner. If you spawn your runner (application server process) using method 1 (direct runner spawn), you should specify the name of your runner. This setting is ignored if you are using spawning method 0.

For example:

```
# Runner name
fgldrunName="myfgldrun.exe"
```

Expected type: string.

Runner Target

Allows you to specify a runner target. If you spawn your runner using method 1 (direct runner spawn), you should specify the target of your runner. This setting is ignored if you are using spawning method 0.

For example:

```
# Start module
fgldrunTarget="main.42r"
```

Expected type: string.

Runner Environment

Allows you to specify environment variables for your runner. If you spawn your runner using method 1 (direct runner spawn), you should specify the environment of your runner. This setting is ignored if you are using spawning method 0.

You can specify up to 20 environment variables (from **fglrunEnv0** to **fglrunEnv19**). Do not use environment variables inside the definition of environment variables:

The following example is correct:

```
fglrunEnv0=" INFORMIXDIR=C:\INFORMIX"
fglrunEnv1=" FGLDIR=C:\INFORMIX\COMPILER"
fglrunEnv2=" PATH=C:\INFORMIX\BIN"
fglrunEnv3=" FGLGUI=2"
fglrunEnv4=" FGLPROFILE=C:\INFORMIX\COMPILER\etc\fglprofile.web"
```

The following example is incorrect:

```
fglrunEnv0=" INFORMIXDIR=C:\INFORMIX"
fglrunEnv1=" FGLDIR=C:\INFORMIX\COMPILER"
fglrunEnv2=" PATH=C:\INFORMIX\BIN"
fglrunEnv3=" FGLGUI=2"
fglrunEnv4=" FGLPROFILE=%FGLDIR%\etc\fglprofile.web"
```

For example:

```
# Environment
# Note: do not use environment variables within the definition of
environment variables.
fglrunEnv0=" "
```

Expected type: string.

Arrays

The following flags enable configuration for DISPLAY ARRAY and INPUT ARRAY:

- **arrayAsButton**: Array as button
- **arrayImage**: Array image

Array as Button

Allows you to use a bitmap as the link to an item in an array. Specify 1 to use a bitmap as a link or 0 to not use a bitmap.

For example:

```
# Array As Button (default : 0)
arrayAsButton=1
```

Define an image with `arrayImage`.

Expected type: integer.

Array Image

Allows you to specify the bitmap used as the link to an item in an array. Use `arrayAsButton` to use a bitmap as the link to an item in an array.

For example:

```
# Array Image (default : "/Cli-HTML/images/bullet1.gif")
arrayImage="/Cli-HTML/images/bullet1.gif"
```

Expected type: string.

Troubleshooting the UNIX Installation

If you have problems installing on UNIX, 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:

```
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:

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

To run the HTML client, type:

```
fglcl
```

If you see a display similar to 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.
Run the **ping** utility to check whether the client can contact the server host. For example:

```
ping 158.58.23.30
```

Try to run **telnet** and connect to the server host from the client.
For example:

```
telnet 158.58.23.301526
```

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 command:

```
netstat -a
```

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

```
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 similar to the following display:

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

```

When you press RETURN, the HTML code for the initial page of the demonstration 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/fglcl.exe?demo">
<HEAD>
<BODY BGCOLOR="#F5F5F5">
IMG SRC=
```

Manual Installation on UNIX

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

These tasks are described in more detail on [pages 10-69 through 10-72](#).

Extracting the Files

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 -xf -
```

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, 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 Windows 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):

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

cp d4gl/Cli-Html/WebServer/cgi-bin/SLS-0250/fglcl.conf
   usr3/httpd/cgi-bin
chmod 644 usr3/httpd/cgi-bin/fglcl.conf
```


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:

- **fglhtml.d** (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):

```
cp /d4gl/Cli-Html/AppServer/bin/SLS-0250/fglhtml.d
   $FGLDIR/bin
chmod 755 $FGLDIR/bin/fglhtml.d

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 the HTML Documentation on the Web Server

The HTML documentation describes the **fglcl.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 UNIX or the Windows NT version of the **phonebook** example in a directory.
2. Run **make** on UNIX or **nmake** on Windows 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 Windows NT Installation

If testing reveals a problem, you can check the HTML client and the HTML server to verify that each is running. For more information, see [“Troubleshooting the UNIX Installation” on page 10-66](#). This section gives differences that apply to Windows NT.

Checking the HTML Client

To set the **QUERY_STRING** environment variable on Windows 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
```

Using the Java Client

In This Chapter	11-3
Introduction	11-3
Programs and Applets	11-4
Swing	11-5
Server-Side Components	11-5
How Dynamic 4GL Uses Java	11-5
Java Client Limitations	11-8
Java Client Security	11-8
Java Client Definitions	11-8
Aliases	11-9
Tag Words and Paths	11-9
Requirements.	11-11
Java Client Web Browser Requirements	11-11
Client Java Applet Viewer Requirements	11-12
Web Server Hardware and Software Requirements	11-13
Dynamic 4GL Application Server Requirements	11-13
Installing the Java Client	11-14
UNIX Installation	11-14
Verifying Required Components	11-15
Running the Shell Script	11-15
Installing on the Web Server	11-18
Installing the Client Component	11-19
Performing Additional Tasks.	11-20
Windows NT Installation	11-20
Automatic Installation	11-20
Manual Installation	11-21
Installing Client Components	11-21

Additional Installation Tasks	11-23
Installing swingall.jar and Setting CLASSPATH on the Client	11-23
Unjarring the cjac.jar file	11-28
Configuring the Servlet Engine for Use with the Java Client	11-29
Verifying Your CLASSPATH Setting on the Web Server.	11-35
Testing the Installation	11-35
Configuring the Java Client	11-37
Editing the cjac.cnf File	11-38
Setting Environment Variables	11-38
Setting Commands and Arguments for Application Execution	11-41
Setting General Parameters Governing CJAC Behavior	11-41
cjac.comm.client.http.requestTimeout	11-42
cjac.comm.client.http.requiredBandwidth	11-42
cjac.comm.client.http.getTimeout	11-42
cjac.comm.server.task.reannounceDelay	11-42
cjac.comm.server.task.startUpTimeout	11-43
cjac.comm.server.tcp.basePort	11-43
cjac.comm.server.tcp.maxConnection	11-43
cjac.comm.server.tcp.portRange	11-43
cjac.comm.server.tcp.reuseDelay	11-44
cjac.setup.check.arg	11-44
cjac.setup.check.enabled	11-44
Sample cjac.cnf file.	11-45
Local and Remote Connections to the Application Server	11-48
Editing the clijava.cnf File	11-49
Changing Colors	11-50
Configuring Interface Elements	11-51
Font Types and Known Font Equivalentents	11-52
Configuring Other Java Applet Elements	11-53
Running an Application with the Java Client	11-54
Creating the HTML Page	11-54
Setting CJA Parameters	11-55
Parameter Settings not Available in clijava.cnf	11-55
Parameter Settings Available in clijava.cnf	11-56
Running the Application	11-56
Java Client Enhancements	11-57

In This Chapter

This chapter describes how to use the Java Client, including how to configure the client and Web server. In this chapter:

- *application* denotes the 4GL application with which you are communicating through a Web browser
- *applet* denotes the Cli Java Applet (CJA), which interacts with the Cli Java Application Connector (CIAC) software to display the 4GL application
- *application server* denotes the computer on which the compiled Dynamic 4GL application is executed by the Dynamic 4GL runner.

This is usually (but not necessarily) the same computer on which the Dynamic 4GL compiler resides.

Introduction

This section gives a brief overview of the Java programming language created by Sun Microsystems. It also describes how Dynamic 4GL uses Java.

Java is an object-oriented programming language with syntax similar to that of C++. The Dynamic 4GL compiler and its associated Dynamic Virtual Machine (DVM) has an architecture similar to that of Java with the Java Virtual Machine (JVM). You compile your Java code (source files with a **.java** extension) into objects described in files with a **.class** extension. There are often many such **.class** files, so they are commonly stored in **.jar** or **.zip** archives.

There is no linker in Java. Instead, whenever a class is required, a set of predefined directories is searched first, followed by each directory or file that is listed in the setting for the **CLASSPATH** environment variable. Java uses **CLASSPATH** the way the operating system uses the **PATH** variable, except that the **CLASSPATH** setting can specify **.jar** and **.zip** archives in addition to directories and files.

After Java source code has been compiled, it is known as *bytecode*. Rather than a machine-dependent code such as C++ generates, Java generates portable code, somewhat like Dynamic 4GL P code. The bytecode can be copied to and used by any platform that has a *Java Virtual Machine* (JVM), which is also sometimes known as the *Java Runtime Environment* (JRE).

The Java development environment is called the *Java Development Kit*, or JDK. Versions for Java and the JDK are often expressed interchangeably, so that when you hear, “My JDK Version is 1.1,” it can mean, “I am using Java Version 1.1.”

Programs and Applets

A Java program consists of a single class which, in turn, references other classes. The class is executed using the Java Virtual Machine, which is named **java**. For example, to execute the program **myjava.class** from the command line, you would type:

```
java myjava.class
```

The **java** command is often embedded in a shell script (on UNIX) or batch file (on Windows) for convenience.

An *applet* is a Java program that is executed inside a Web browser. To execute an applet, you load a Web page that includes a special HTML tag similar to the tag used to load an image. The browser then downloads the applet and executes it using the JVM. Because the applet is stored on the Web server and only downloaded as it is being executed, deployment and maintenance of the applet is much simpler than deploying an equivalent Java program to many sites and then maintaining all the sites.

Swing

The original Java user interface was written in a set of classes called *Abstract Window Toolkit* or AWT. AWT relied heavily on native libraries, and therefore had portability issues. A new implementation of the user interface was included in the *Java Foundation Classes* (JFC), introduced in 1997, through a set of new components called *Swing*. Swing is written entirely in Java and does not require extra system libraries, avoiding portability problems among different environments.

The Swing components are not part of Java Version 1.1, but you can use them with that version by installing them separately. Although most Web browsers do not support Swing by default, you can usually add support by adding the **jar** file for the Swing package to the **CLASSPATH** setting.

Server-Side Components

Java was designed to efficiently handle connectivity across a network. You can write Java programs to behave as servers or to interact with other network components, such as Web servers.

To extend a Web server, you can use the *Common Gateway Interface* (CGI). This technology is used by the Dynamic 4GL HTML client (see [Chapter 10, “Using the HTML Client,”](#) for more information.) CGI is less efficient with Java, however, because CGI starts a new process every time it receives a new call. With Java, this means starting a new JVM and then a Java program with each call.

Instead, Dynamic 4GL uses *servlets*, created using the *Java Servlet Development Kit* (JSDK). The servlets reside on one virtual machine and persist there, lowering startup time and transmission bandwidth and saving memory.

How Dynamic 4GL Uses Java

To interact with Java, the Dynamic 4GL runner sends the output of the Dynamic 4GL Virtual Machine to a servlet instead of to WTK (on Windows) or to **fglX11d** (on UNIX). The servlet, called the *Cli Java Application Connector* (CIAC), handles communication between the client (the Web browser) and the runner.

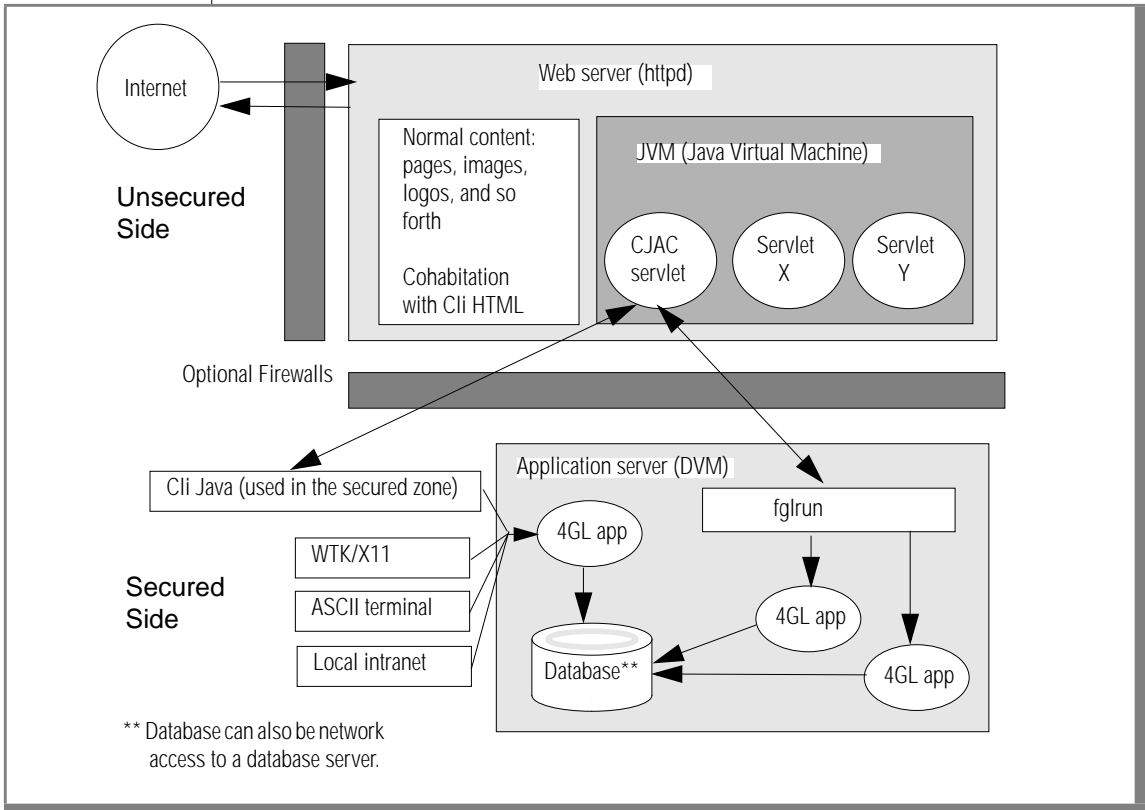
The interface to the user is handled by the *Cli Java Applet* (CJA), which communicates with CJAC to display information and support user input.

The detailed sequence of operations is as follows:

- The Web browser (or appletviewer) initiates a request.
The browser (or appletviewer) must be compliant with JDK 1.1.3 or later and should support the Java Foundation Classes (Swing) Version 1.1. Swing must be installed separately.
- The Web server spawns and communicates with CJAC.
The Web server must be able to run servlets.
- The Java Application Connector communicates with a Dynamic Virtual Machine (DVM) that handles and interprets the P code.
The CJAC servlet starts a DVM that communicates with the applet.

The following figure shows the complete environment for the Java Client. You can run the Java Client through a firewall, as shown in [Figure 11-1](#).

Figure 11-1
Java Client Firewall Architecture



Java Client Limitations

When using the Java Client, be aware of the following limitations:

- With 4GL, each character entered into a field is analyzed and transmitted to the runner.
With local editing on Dynamic 4GL, the entire field is analyzed at once and transmitted to the runner.
- Avoid interactive system calls.
The Java Client does not include terminal emulation software (the Windows Client, however, does include such software).
- You cannot use DDE (even on Windows).
- A Java applet cannot access the local system.
This means you cannot use systems calls such as **rcp** or **winexec**, and you cannot use local storage.
- You must use **.gif** or **.jpg** images.
You cannot use **.bmp** images with the Java Client.

Java Client Security

Security is an important concern for software that runs on the Web. The Java Client software does not include specific security features, so you are encouraged to use existing security measures and protocols, including firewalls and the secure socket layer (SSL) protocol in addition to preserving the security features in your original application. If you are using an extranet or the Internet, you might consider instituting a login and password scheme.

Java Client Definitions

Before installing the Java Client, review the following definitions. These definitions are used throughout the installation.

Aliases

An *alias* (or virtual directory) is a name you define as a substitute for a real path name. An alias called **clijava** can point to any directory on your Web server. The advantage of using an alias is that the full path is invisible to the user.

For example, if your document root directory is **/usr/htdocs**, when you create a directory **clijava** in **/usr/htdocs** (the full path is then **/usr/htdocs/clijava**), it will be seen by a user as **clijava**.

For more information on aliases and virtual directories, refer to your Web server documentation.

Tag Words and Paths

The following table lists tag words and paths that are used in the descriptions of how to install and configure the Java Client.

Tag Word	Explanation	Example
<i>web_server</i>	The Web server IP address. In most cases it can be replaced by the Web server name.	10.0.0.1
<i>web_server_port</i>	The port your Web server is listening to. In most cases it is 80.	80, 8080, 8081...
<i>IP address</i>	An IP address in the aaa.bbb.ccc.ddd format, where aaa , bbb , ccc and ddd are integers between 0 and 255.	10.0.0.3
<i>web_server_servlet_dir</i>	The Web server servlets directory, seen as a UNIX or a Windows NT path.	/usr/local/apache/share/servlets or c:\inetpub\wwwroot\servlets

(1 of 2)

Tag Word	Explanation	Example
<i>web_server_servlet_dir_alias</i>	<p>The servlets directory alias for the Web server: it is a virtual path from the WWW root directory to <i>web_server_servlet_dir</i>.</p> <p>In this release, the alias must be /servlets/. The applet (CJA) will contact the servlet (CJAC) through an http request: <code>http://<web server>:<web server port>/servlets/cjac</code> and will fail if there is no answer.</p>	<p>servlets, slets...</p>
<i>web_server_clijava_dir</i>	<p>The directory where you want to install your "call" to Cli Java. This directory is a subdirectory of your document directory or an alias that points to it. This directory is a path on both UNIX and Windows NT.</p>	<p>/usr/local/apache/share/htdocs/clijava</p>
<i>web_server_clijava_dir_alias</i>	<p>The Web alias directory (or virtual directory) where you want to install your "call" to Cli Java.</p>	<p>clijava...</p>

(2 of 2)

Requirements

This section covers the hardware and software requirements for installing the Java Client.

Java Client Web Browser Requirements

The following table summarizes the supported hardware and software for the Java Client Web browser. The Browser or Java AppletViewer must support JDK Version 1.1.

Platform	Version	Hardware	Software
UNIX	Various versions	32 megabytes of RAM (64 megabytes recommended)	Netscape Communicator 4.5 with Java Foundation Classes (Swing) 1.1 Microsoft Internet Explorer 4.x with Java Foundation Classes (Swing) 1.1
Windows	Intel	32 megabytes of RAM (64 megabytes recommended) Intel or compatible Pentium class CPU at 133 MHz (200 MHz recommended) 30 megabytes for the browser, 12 megabytes for the Java Plug-In, and 1 megabyte for the applet	Netscape Communicator 4.5 with Java Foundation Classes (Swing) 1.1 Microsoft Internet Explorer 4.x with Java Foundation Classes (Swing) 1.1 Instead of the Swing Java Classes, you could install the Sun Java Plug-In. The Sun Plug-in is only available for Solaris, Windows 9x, and Windows NT 4.0.
Mac OS	PowerPC	32 megabytes of RAM (64 megabytes recommended) PowerPC 603e at 200 MHz (PowerPC G3 at 233 MHz recommended) 20 megabytes for the MRJ (Mac OS Runtime for Java) and 1 megabyte for the applet	Apple MRJ (Mac OS Runtime for Java) 2.1 MacOS 8.1 or later Microsoft Internet Explorer 4.0



Important: Browser size can vary widely from one operating system to the other and depends upon the options the user selects when installing the browser.

Client Java Applet Viewer Requirements

The following table summarizes the supported hardware and software for the Java Client Applet Viewer.

Platform	Version	Hardware	Software
UNIX	various versions	32 megabytes of RAM (64 megabytes recommended)	Java Runtime Environment Java Foundation Classes (Swing) 1.1
Windows	Intel	32 megabytes of RAM (64 megabytes recommended) Intel or compatible Pentium class CPU at 133 MHz (200 MHz recommended) 12 megabytes for the Java Plug-In and 1 megabyte for the applet	Java Runtime Environment 1.1.3 (Java Runtime Environment or Java Development Kit 1.1.7 recommended) Java Foundation Classes (Swing) 1.1
Mac OS	PowerPC	32 megabytes of RAM (64 megabytes recommended) PowerPC 603e at 200 MHz (PowerPC G3 at 233 MHz recommended) 20 megabytes for the MRJ (Mac OS Runtime for Java) and 1 megabyte for the applet	Apple MRJ (Mac OS Runtime for Java) 2.1 MacOS 8.1 or later

Web Server Hardware and Software Requirements

The following table summarizes the supported hardware and software for the Java Web Server. You need a Web server that supports Java servlets, such as Sun's Java Web Server, Apache with Apache JServ, IIS with JRun, or Netscape with JRun.

Platform	Version	Hardware	Software
UNIX	various versions	32 megabytes of RAM (64 megabytes recommended) 5 megabytes for the HTTP server, 42 megabytes for the Java Development Kit (Java Run Time Environment), and 1 megabyte for the servlet	Web server software that supports servlets and is compliant with JSDK 2.0 JDK/JRE 1.1.3 or later
Windows NT	Intel	32 megabytes of RAM (64 megabytes recommended) Intel or compatible Pentium class CPU at 133 MHz (200 MHz recommended) 5 megabytes for the HTTP server, 5 megabytes for the Java Run Time Environment, and 1 megabyte for the servlet	Web server software that supports servlets and is compliant with JSDK 2.0 JDK/JRE 1.1.3 or later



Important: Web server (HTTP server) size can vary widely from one operating system to the other and depends upon the options the administrator selects when installing the server.

Dynamic 4GL Application Server Requirements

The following table summarizes requirements for the application server.

Software Requirements	Hardware Requirements
Dynamic 4GL runtime (DVM) version 3.x (or later)	For hardware requirements and information about installing the compiler, refer to Chapter 2 .

Installing the Java Client

After you configure your Web server for servlets, you can install the Java Client. You can install the Java Client on UNIX or Windows NT.

These instructions assume that you are installing the Java Client on a computer that acts as both application server and Web server. The client (user-interface) portion of the architecture is handled through the Swing classes you install in your Web browser. (Installing Swing is described in [“Installing swingall.jar” on page 11-24.](#))

UNIX Installation

UNIX installation includes:

- Verifying required components
- Running the shell script
- Installing the client component (**swingall.jar**)
- Performing additional installation tasks

Verifying Required Components

To use the Java Client, you need the proper software configured and installed. Verify that you have all the following components:

- Client with TCP/IP access and a Java enabled browser (or Java Applet Viewer)
The browser (or Applet Viewer) must be compliant with JDK 1.1.3 or later and should support JFC (Swing) Version 1.1.
- Web server that supports Java servlets, such as Sun's Java WebServer, Apache with Apache JServ, IIS with JRun, or Netscape servers
- Cli Java Application Connector (CJAC)
- Application Server
- Compiled Dynamic 4GL application and runner

Running the Shell Script

The installation shell script is named **cljava-all-version-allos.sh** where *version* is the version number of the software.

To install the Java Client

1. Type the following command:

```
sh clijava-all-0.90.1el-allos.sh -i
```

The following message appears:

```
#####
Informix Dynamic 4GL Java Client <version> for <os>
#####

Identifying your system..... <os>
Looking for df command..... Available
SHELL is..... /bin/sh
Looking for Unix commands..... Ok
Looking for shell commands..... Ok
Looking for ln -s..... Ok
Current user is ..... Not SUPERUSER

WARNING -----
With this user some administration operations will be skipped.
-----
```

The following prompt appears and offers to continue the installation even though you are not the root user:

```
Do you want to continue to install the product?
Options: ( [Y]es | [N]o | [C]ancel | ?)
Default: [Y]
```

You need not be root to install the Java Client Web server. However, as most Web servers are installed as root, it might be better to be the same user as when the Web server was installed. The installation does copy some files to the Web server and you will need appropriate access rights. For these access rights, see your Web server documentation.

2. Type Y to continue the installation.

The following list of installation options appears:

```
Checking command chmod for this user..... Ok
Looking for clijava.tgz in /tmp/decomp..... Ok

#####
Welcome to Informix Dynamic 4GL Java Client installation script
#####

1 --- Cli Java Application Server components
2 --- Cli Java Web Server components
3 --- Cli Java Application Server & Cli Java Web Server components
4 --- Cli Java Client components
5 --- Cli Java User Manual only

Options: ( VALUE | [C]ancel)
Default: [1]
```

3. Select an installation option.

For instance, to install both the Web server and application server components, select 3. The following messages appear:

```
Free disk space in /tmp/install..... 304048 blocks
Preparing Cli Java installation package..... Ok
```

4. Several components need to be installed on the application server. The installation shell script uses your **FGLDIR** setting as the default directory. If you choose a directory other than **FGLDIR**, you must place the components in the proper **FGLDIR** directories after installation.

```
#####
Application Server - Installation
#####

You selected Cli Java Application Server components installation,
Where do you want to install them (usually $FGLDIR)
Options: ( VALUE | [C]ancel | ?)
Default: [/usr/fgl2c
Free disk space in /usr/fgl2c..... 1008698 blocks
Java Client Application Server package..... Ok
```

You are now finished installing components on the application server and can begin installing components on the Web server.

The Cli Java Applet is the first component. The following message appears:

```
#####
Web Server - Applet Installation
#####
```

You selected Cli Java Web Server components installation Part 1/2.

Installing on the Web Server

1. Two main groups of files are installed on the Web server:
 - Some files go into the documents directory of the Web server (often named *htdocs*). Included among these files are the CJA applet, downloaded by a browser connecting to the Web server, and the sample HTML pages that call demonstration applications.
 - The remaining files, including the CJAC servlet, go into the servlets directory.

The following prompt appears for the directory where you want to install the first group of files:

```
Enter your HTML documents home directory (htdocs) ?
Options: ( VALUE | [C]ancel | ?)
Default: [/usr/local/apache/htdocs]
```

2. Enter the root directory of your documents directory on your Web server.

This value will be used to define *web_server_clijava_dir*. If possible, use the default value as it will later make configuring the server easier.

```
CJA and related files will be installed in:
[/usr/local/apache/htdocs/clijava]
Do you agree ?
Options: ( [Y]es | [N]o | [C]ancel | ?)
Default: [Y]
Free disk space in ~ocal/apache/htdocs/clijava.... 293298 blocks
Cli Java CJA package - Part 1/2..... Ok
```

3. Next, you receive the following prompt for the location in which to install the CJAC servlet and related files:

```
#####
Web Server - Servlet Installation
#####
Enter your servlet home directory (servlets) ?
Options: ( VALUE | [C]ancel | ?)
Default: [/usr/local/apache/servlets]
```

This value is *web_server_servlet_dir*.

```
CJAC files will be installed in:
[/tmp/servlets]
Do you agree ?
Options: ( [Y]es | [N]o | [C]ancel | ?)
Default: [Y]
```

```
Free disk space in /usr/local/apache/servlets.... 292956 blocks
Cli Java CJAC package - Part 2/2..... Ok
```

```
#####
User Manual - Installation
#####
Would you like to install the HTML version of the Cli Java
documentation?
Options: ( [Y]es | [N]o | [C]ancel | ?)
Default: [Y]
You selected Cli Java HTML documentation installation.
Do you want to install it in the following directory ?
[/usr/local/apache/htdocs/cli/java/manual]
Options: ( [Y]es | [N]o | [C]ancel | ?)
Default: [Y]
Free disk space in ~ache/htdocs/cli/java/manual.... 292045 blocks
Cli Java User Manual Insatallation..... Ok
You can now read the User Manual with your browser, open it with the
following URL:
```

```
file:///usr/local/apache/htdocs/cli/java/manual/index.html
```

You have now completed the installation and will see the following banner:

```
#####
End of the installation process
#####
```

Installing the Client Component

The Java Client package includes a **swingall.jar** file, which must be placed on the client computer. You can choose to install this component on the Web server computer and later transfer the file to the client, or you can run the installation shell script on the client computer itself (this is Option 4 in the installation shell script). For more information about installing **swingall.jar** on the client, see [“Installing swingall.jar and Setting CLASSPATH on the Client”](#) on page 11-23.

Performing Additional Tasks

To continue the installation, see [“Additional Installation Tasks”](#) on page 11-23.

Windows NT Installation

Before beginning the installation, verify that you have the following required components to run the Java Client:

- Client with TCP/IP access and a Java enabled browser (or Java Applet Viewer).
The browser (or Applet Viewer) must be compliant with JDK 1.1.3 or later and should support the JFC (Swing) Version 1.1.
- Web server that can support Java servlets and is JSDK 2.0 compliant, such as Apache with Apache JServ, IIS, or Netscape with JRun.
- Application server
- Compiled Dynamic 4GL application and runner.

Windows NT offers both automatic and manual installation.

Automatic Installation

Automatic installation downloads all the files and handles installation of all the components. The sections that follow ([“Downloading Installation Files”](#) through [“Installing Client Components”](#)) document manual installation. Read those sections for the steps required for successful installation.

If you are installing the Java Client using automatic installation, the installation wizard will guide you through these steps. Run the installation wizard executable in the Java Client folder within the Clients directory on your product CD.

Be prepared to give destination directories for the various parts of the Java Client architecture. For example:

- The application server package should be placed in the directory that the **FGLDIR** environment variable indicates.
- The Web server applet package should be placed in *web_server_cljava_dir*.
- The Web server servlet package should be placed in *web_server_servlet_dir*.
- The documentation package can be placed in the documents directory of your Web server.

Manual Installation

Manual installation includes the following steps:

- Downloading installation files
- Installing application server components
- Installing Web server components
- Installing client components
- Installing HTML documentation

Installing Client Components

Installation includes the steps in the following sections.

Downloading Installation Files

This step is necessary only if you are installing the Java Client manually. Copy the files from the CD to a folder on your hard drive. Then follow the steps in the following sections to place the Java Client components in the proper directories.

Installing Application Server Components

Several Java Client components must be placed in your **FGLDIR** directory on your application server. These components are included in the **appserver.tgz** file in your Java Client installation directory. They include:

- **etc\clijava.cnf**
- **etc\clijava.res**
- **msg\clijava.iem**
- **src\clijava.msg**
- **clijava**
- **clijava\release**
- **clijava\release\k**
- **release.txt**

To install these files, unzip the contents of **appserver.tgz** and extract the files to your **FGLDIR** directory.

Installing Web Server Components

Two groupings of files must be placed within your Web server. Some files belong in the documents directory of the Web server. Included among these files are the CJA applet downloaded by browsers connecting to the Web server and the sample HTML pages that call demonstration applications. These files are included in the **webserverapplet.tgz** file in your Java Client installation directory.

The remaining files, including the CJAC servlet, go into the servlets directory. These files are included in the **webserverservlet.tgz** file in your Java Client installation directory.

To install the contents of **webserverapplet.tgz**, unzip the file and extract the contents to your *web_server_clijava_dir* directory. This directory should reside within your Web server documents directory. An example *web_server_clijava_dir* might be **C:\Inetpub\wwwroot\docs\clijava**.

To install the contents of **webserverservlet.tgz**, unzip the file and extract the contents to your *web_server_servlet_dir*. An example *web_server_servlet_dir* might be **C:\Inetpub\wwwroot\servlets**.

Installing Client Components

The Java Client package includes a **swingall.jar** file, which must be placed on the client computer. This file is included in the **client.tgz** file in your Java Client installation directory. For more information about installing **swingall.jar** on the client, see [“Installing swingall.jar and Setting CLASSPATH on the Client”](#) on page 11-23.

Installing the HTML Documentation

To install the HTML version of the Java Client documentation, extract the contents of the **manual.tgz** file into a directory accessible to your Web server.

Additional Installation Tasks

Additional tasks include the following:

- Installing **swingall.jar** on the client and setting the client **CLASSPATH** environment variable
- Unjarring the **cjac.jar** file
- Configuring the servlet engine for use with the Java Client
- Verifying your **CLASSPATH** setting
- Testing the installation

Installing swingall.jar and Setting CLASSPATH on the Client

The client computer displays the Java Client using a Java enabled Web browser or Java applet viewer. To accomplish this, the client computer must have the Swing library installed. The following sections describe how to install the Swing library.

First, to verify that your client can support the Java Client, run the Client Detection Wizard. The wizard checks the operating system, browser version, and whether the browser has the appropriate Java software installed.

The Client Detection Wizard Web page is included in *web-server-clijava-dir* (see [“Tag Words and Paths”](#) on page 11-9.) The Web page is called **res_clijava_detection_wizard.html**. Display the page in a Web browser on the client computer to run the Detection Wizard.



In addition, to install Swing Java classes, you must set the **CLASSPATH** environment variable. The steps for setting this environment variable differ depending upon the platform. See [“Setting CLASSPATH” on page 11-25](#) for more information.

***Tip:** If you are using a Web browser, you can install the Sun Java Plug-in instead of the Swing Java classes. However, it is recommended you install the Swing Java classes. If you want to install the Sun Java Plug-in, refer to the Sun Web site for more information.*

Installing swingall.jar

The **swingall.jar** file is included with Dynamic 4GL Java Client package; you choose to install it as part of the installation process. After installation, you must copy this file to the client computer. Where you copy the file depends on your environment, as follows:

- **Netscape Communicator.** You need Version 4.5 with full support of JDK 1.1. If you have installed Netscape Communicator, install the Swing package in:

```
C:\ProgramFiles\Netscape\Communicator\Program\java\classes
```

- **Microsoft Internet Explorer.** Copy the **swingall.jar** file to **C:\swing-1.1**. Avoid copying the file to a directory path that contains spaces, such as **C:\Program Files\Swing-1.1**. Some programs have problems recognizing spaces in the path name.
- **UNIX.** Copy the **swingall.jar** file to a directory. If possible, copy the file to your **\$FGLDIR/clijava/lib** directory.

On computers that do not have a compiler or runtime system, you can select any directory. For instance:

```
/usr/local/lib/java/swingall.jar.
```

It might be necessary to restart your browser.

- **Macintosh.** Download the Swing installer from the Sun Java Web site. Double-click the **Swing 11-Install** icon to begin the installation. The installation wizard guides you through the installation steps. At the appropriate prompt, choose the **Runtime Only** option.

Setting CLASSPATH

After installing the Swing Java classes, set the **CLASSPATH** environment variable to include **swingall.jar**. The following directions describe how to set the environment variable for UNIX, Windows 9x, Windows NT, and Macintosh clients.

UNIX

Set the **CLASSPATH** environment variable as follows:

```
CLASSPATH=$FGLDIR/clijava/lib/swingall.jar:$CLASSPATH
export CLASSPATH
```

In order for **CLASSPATH** to be set appropriately each time a user starts an application, you should place the **CLASSPATH** entry in the generic **/etc/fglprofile** file or in each user's **.fglprofile** file.

You must restart your browser for the setting to take effect.

Windows 9x

Edit your **c:\autoexec.bat** file and add the line:

```
SET CLASSPATH=C:\swing-1.1\swingall.jar;%CLASSPATH%
```

If you copied the **swingall.jar** file to a different directory, substitute the appropriate directory for **C:\swing-1.1**.

When you are finished, reboot your computer for the changes to take effect.

Windows NT

1. Start the **Control Panel**.
2. Double-click the **System** icon.
The System Properties dialog box appears.
3. Click the **Environment** tab.

4. Select the **ClassPath** environment variable in the System Variables list box.

If the **ClassPath** environment variable does not exist, click any line in the System Variables list box.

The current user must have the appropriate rights to set environment variables. The environment variable needs to be set for all users (not just for the current user).

5. In the Variable text box, enter **CLASSPATH**.
6. In the value text box, enter **C:\swing-1.1\swingall.jar**.

If you copied the **swingall.jar** file in another directory, change **C:\swing-1.1** to the appropriate file location.

If you have other **CLASSPATH** entries, separate them with a semicolon (;).

7. Click **Set**.
8. Click **OK**.
9. Restart Netscape Navigator or Internet Explorer, if necessary.

Macintosh

Follow these steps to add the Swing classes to the Internet Explorer Class Paths settings:

1. Start Internet Explorer.
It loads and you should get either a home page or a blank screen (depending on your users settings). Select the Edit menu and then select Preferences.
The Internet Explorer Preferences dialog box appears.
2. Click Java in the Web Browser settings list box.
The Java settings appear with the following panes: Java Options, Class Paths, Security Options
3. In the Java Options pane, check the Enable Java checkbox and then select Apple MRJ from the Java virtual machine list box.
Apple MRJ is more compliant to Sun's Java standard.
4. In the Class Paths pane, click **Add**.
A Select Item dialog box appears.

5. Select the **swingall.jar** file.
The location depends on where you installed the folder. By default, it is **<volume>:Applications:Swing-1.1:swingall.jar**.
On the Macintosh, path separators are colons (:).
6. Click **Select**.
A prompt appears that reminds you to restart Internet Explorer to have the changes take effect. You should restart your browser before using your new class path.
7. Click **OK**.
8. Check the Security Options panel to ensure that the following conditions are set:
 - Byte-code verification:** Check All Code
 - Network access:** Applet Host Access
9. Restart your browser.

Testing Your Swing Installation

An HTML page is supplied to let you test your Swing installation. To perform the test, navigate in your browser to:

```
http://web_server_dir:web_server_port/clijava/res_test_swing.html
```

You should see the page shown in [Figure 11-2](#).

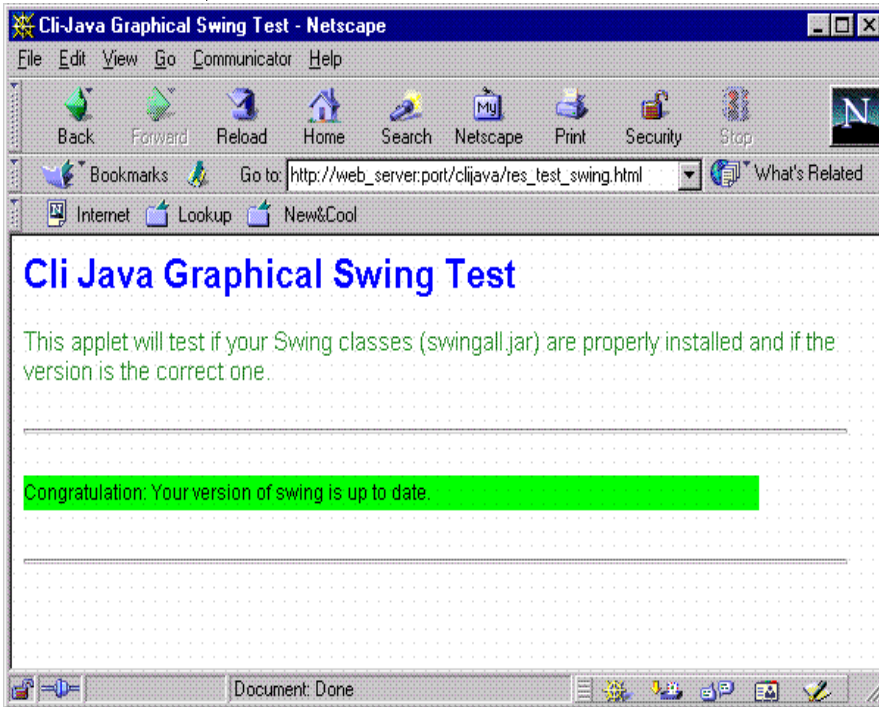


Figure 11-2
CLI-Java Graphical
Swing Test Page

Unjarring the cjac.jar file

The **`cjac.jar`** file that you placed in the servlets directory must be unjarred to allow the Java Client to access the files (see [“Running the Shell Script”](#) on page 11-15 for UNIX or [“Installing Web Server Components”](#) on page 11-22 for Windows).



Tip: The `web_server_servlet_dir_alias` must be reached through the alias “`servlets`,” however, this need not be the only alias. When using servlets and applets, remember that names are case sensitive.

Follow these steps to unjar the file:

1. Navigate to `web_server_servlet_dir` and enter:

```
jar -xvf cjac.jar
```

This command is available on both UNIX and Windows NT.

If **jar** is not a command of your system, check your JDK or JRE installation and the current environment. This command is part of the standard JDK or JRE package.

This step can be skipped if you can set **CLASSPATH** using directives in your Web server configuration files, but it is not recommended that you do so. Refer to your Web server documentation for more information. If you edit a configuration file, you might need to restart the Web server.

After you unjar the **cjac.jar** file, you should see the following directory structure underneath `web_server_servlet_dir`:

```
com
lib
fglTestServlet.class
cjac.jar
META-INF
```

2. You can now safely remove the **cjac.jar** file:

```
rm web_server_servlet_dir/cjac.jar (Unix)
del web_server_servlet_dir\cjac.jar (Windows NT)
```

Configuring the Servlet Engine for Use with the Java Client

If you have a properly installed Web server and servlet engine, you should already be able to run basic servlets. Your servlet engine often provides example servlets for you to verify this setup. If your Web server is not yet capable of running JSDK 2.0 servlets, refer to your Web server or servlet engine documentation for assistance.

Additional steps must be taken to enable your servlet engine to recognize and interact with components of the Java Client. The way in which these modifications are made differ depending on the Web server and servlet engine you are using. A general explanation of what must be done and examples on a few of the major platforms are in the following sections.

Servlet Mapping

When the user requests an application via the browser, the CJA applet makes a call to the Cli Java Application Connector (CJAC) servlet. CJA launches CJAC using the URL `/servlets/`. Your Web server environment must be capable of recognizing such calls and invoking the servlet engine when they are received.

This allows the servlet engine to run a target file called with the name `/servlets/` as a servlet. This is often referred to as *mapping*. Some servlet engines are mapped to `/servlet/` (not `/servlets/`) by default. Therefore, it might be necessary to configure the Web server to recognize calls to `/servlets/` as well. This is often done by *assigning a style*. An example using Netscape Enterprise Server appears in [“Netscape with JRun on UNIX” on page 11-31](#).

Servlet Aliases

In addition, you must create two servlet aliases so that calls placed to the alias will call the target servlet. Your Web server or servlet engine should have a facility for adding servlet aliases.

The first alias is for the example servlet **fglTestServlet**. This servlet should be located in your servlets directory upon installation of **cjac.jar**.

fglTestServlet is not part of the functional product but is provided to allow you to verify a working environment and to illustrate the concept of servlet aliases. You will need to create an alias for this servlet as follows:

Alias Name	Class Name
TestServlet	fglTestServlet

This alias enables the Web server to interpret:

```
http://myserver/servlets/TestServlet
```

as a call to invoke the servlet **fglTestServlet**.

All servlets should reside in or underneath the servlets directory. Because **fglTestServlet** is located in the servlets directory itself (for example, **Apache/servlet_directory/fglTestServlet**), you could invoke this servlet by pointing to:

```
http://myserver/servlets/fglTestServlet
```

However, it might not always be convenient or desirable to call the servlet by its filename.

The CJA calls the **CJAC.class** file using the alias **cjac**. The **CJAC.class** file resides in the following directory after you unjar the **cjac.jar** file:

```
servlet_directory/com/informix/communication
```

You must create an alias that lets the servlet engine know to look for **cjac** in this directory structure either within the **.jar** file or on the file system under the servlets directory:

Alias Name	Class Name
cjac	com.informix.communication.CJAC

The following examples are for specific Web server environments.

Apache with Apache Jserv on UNIX

The invocation of servlets is handled by the **jserv.properties** file, which should already be configured to run servlets appropriately.

To add the needed aliases, find the **servlet.properties** file, typically located in the directory where Apache JServ is installed (for example, in **Apache/Apache Jserv**). Add the following entries:

```
servlet.cjac.code=com.fourjs.communication.CJAC
servlet.TestServlet.code=fglTestServlet
```

Netscape with JRun on UNIX

Use the JRun administration tool to configure your servlet engine for use with Java Client components.

To add the mapping to /servlets/

1. Start JRun administration.
2. Select **jse**.
3. Click the Service Config button.
4. Select the **Mappings** tab.
5. Click **Add**.
6. Add the following entries and click Save:
Virtual Path/Extension Servlet Invoked
Servlet Invoker

To add the aliases

1. Start JRun administration.
2. Select **jse**.
3. Click Service Config.
4. Select the Aliases tab.
5. Click **Add**.
6. Add the following entries and click **Save**:
Name Class Name
TestServlet fglTestServlet
cjac com.informix.communication.CJAC

These servlets need not be pre-loaded.

When using JRun as your servlet engine, your servlets directory need not be located within JRun. The file **jrun.properties**, located in **/JRun_install_directory/jsm-default/services/jse/properties**, allows you to direct JRun to look for servlets in directories other than **/JRun/servlets** and **/JRun/jsm-default/services/jse/servlets**, which are the default locations.

Add a pointer to your Web server servlets directory to the `servletdir` setting. For example, you might modify the `servletdir` entry as follows:

```
servletdir=/usr/jrun/jsm-default/services/jse/servlets,/jrun/
servlets,/usr/Apache/servlets
```

where *servlets* is your main servlets directory (the location in which **cjac.jar** was installed). This can also be done using the JRun administration utility as follows:

1. Start JRun administration.
2. Select **jse**.
3. Click Service Config.
4. Select the General tab.
5. Add your servlets directory to the **Default Servlets Dir** entry.
6. Click **Save**.

For example:

```
C:/JRun/jsm-default/services/jse/servlets,C:/
JRun/servlets,C:/Inetpub/wwwroot/servlets
```

where **C:/Inetpub/wwwroot** is your Web server root directory and **servlets** is your servlets directory.

You must restart JRun for the changes to take effect.

If it has not already been done, it might also be necessary to *assign a style* within your Netscape Enterprise Server, as follows:

1. Run the Netscape Administration Server.
2. Click **Server Preferences** in the toolbar.
3. Click **View Server Settings** in the side panel.
4. Click **Configuration Styles** in the toolbar.
5. Click **Assign Style** in the side panel.

6. Fill in the requested field with `servlets/*` and choose **JRun** as the style.
7. Click **OK** and **Apply**.

For example:

```
:8090/      servlets/*  
  
Style:      JRun
```

Sun Java Web Server on Windows NT

Configuration for the Sun Java Web server is done using an applet in your browser. Call your Web server on port 9090 which is the JavaWebServer default administration port (you can change this).

When you connect to the administration port with your browser, you will need to provide a login and password. The initial default will be *admin* for both fields.

To add mapping to /servlets/

1. Select **Web Service** from the JavaWebServer Services menu.
2. Click **Manage**.
3. Click Servlet **Aliases** in the directory tree under Setup.
4. Click **Add** and add the following entries:

Alias	Servlet Invoked
/servlets/	invoker

5. Click **Save**.

To add aliases for fglTestServlet and CJAC

1. Click Servlets in the toolbar.
2. Select **Add** from the directory tree and add the following entries:

```
ServletName: TestServlet  
ServletClass: fglTestServlet
```

3. Select **No** in the Bean Servlet box.

4. Click **Add**.
You see another screen with Configuration and Properties tabs. You need not make further changes.
5. Click **Load** to complete the creation of the alias.
6. Repeat for CJAC, adding the following entries:

```
ServletName: cjac
ServletClass: com.informix.communication.CJAC
```

Verifying Your CLASSPATH Setting on the Web Server

If your Web server was capable of running servlets before you began the installation, your class path should already include the correct settings. At a minimum, your settings must include pointers to:

- the JSDK classes (**jsdk.jar**).
- the JDK classes and source files.
- the classes specific to your servlet engine (for example: **ApacheJServ.jar**).
- the Swing classes, if your client computer is also your Web server (**swingall.jar**).

A sample setting on Windows NT follows:

```
C:\Jsdk2.0\lib\jsdk.jar;C:\jdk1.1.8\lib\classes.zip;
C:\jdk1.1.8\src;C:\JRun\lib;C:\swing-1.1\swingall.jar
```

Depending on your Web server environment, you can set **CLASSPATH** in several ways. Refer to your Web server and servlet engine documentation for more information.

Testing the Installation

After you perform the above steps, you should be able to run servlets from your *web_server_servlet_dir*, including CJAC. Testing your installation will involve calling both the provided `fglTestServlet` and the CJAC servlet.

To test this installation, enter the following URL into your browser:

```
http://web_server:web_server_port/servlets/TestServlet
```

You should see the HTML page shown in [Figure 11-3](#).

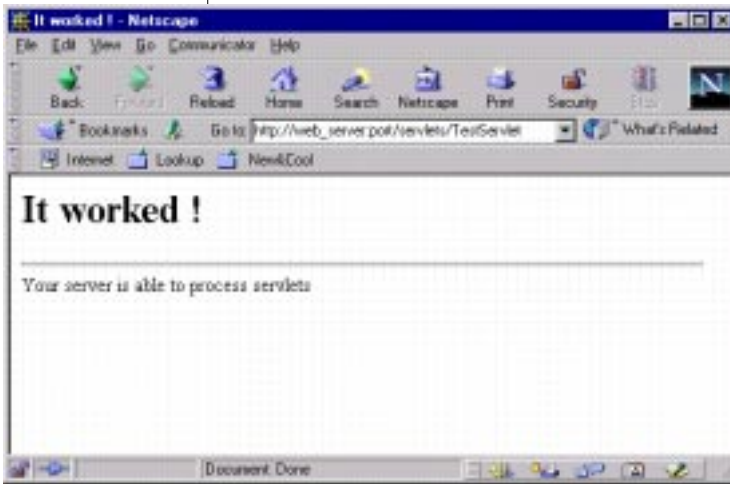


Figure 11-3
*Successful
Servlet Test
Results Page*

If you instead see a page similar to one shown in [Figure 11-4](#), it is likely that **CLASSPATH** is not properly set. Refer to your Web server or servlet engine documentation for more information. The files not found by your Web server often provide clues as to what components are missing from your **CLASSPATH** setting.



Figure 11-4
*Servlet Test
Error Page*

To test the CJAC servlet, enter the following URL into your browser:

```
http://web_server:web_server_port/servlets/cjac?TEST
```

This entry calls the CJAC servlet with the parameter *TEST*. The **cjac.cnf** file, located in **web_server_servlet_dir/lib**, directs cjac to display the file **res_installation_check.html**, located in the same directory.

You should see the HTML page shown in [Figure 11-5](#).

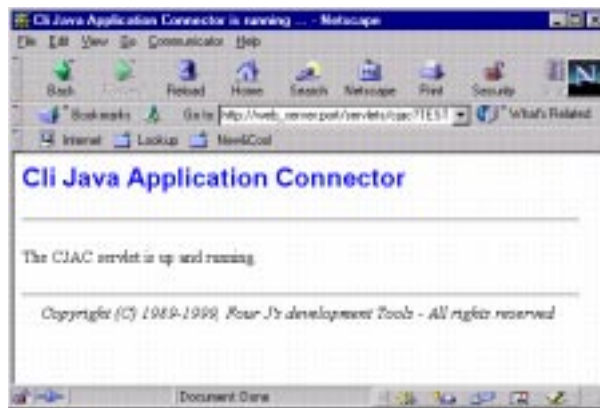


Figure 11-5
Successful CJAC
Servlet Test
Results Page

If you do not see this page, check your **CLASSPATH** settings.

You might also verify that the **cjac.cnf** file is located in your **web_server_servlets_dir/lib** directory.

Configuring the Java Client

Before running a Dynamic 4GL application for display as a Java applet, you must specify an appropriate environment in which to run the application and supply CJAC with information about starting the application. In addition, you must configure your environment to handle applets.

These steps involve modifications to two files included in the Java Client package:

- **cjac.cnf**
- **clijava.cnf**

Editing the *cjac.cnf* File

The *cjac.cnf* file is used by the Cli Java Application Connector (CJAC) to determine the proper environment and set of commands to use to execute a called application. It is located in the ***web_server_servlet_dir/lib*** directory.

The ***cjac.cnf*** file is first read when CJAC is launched (at application startup), and subsequently every 10 seconds. Changes do not take effect, however, until the 4GL application is restarted.

In general, configuration using ***cjac.cnf*** involves three tasks:

- Setting environment variables
- Specifying commands and arguments for application execution
- Setting general parameters governing CJAC behavior

Setting Environment Variables

When CJAC runs Dynamic 4GL locally, you must set environment variables to point to your local environment. A list of variables to set for both UNIX and Windows NT systems appears on [p. 11-39](#).

You must either set all the necessary variables (sometimes with the default values) or comment out those that do not apply using a '#' character.

For more information about environment variables, see [Appendix A](#) and the *INFORMIX-4GL Reference Manual*.

You can set environment variables either as defaults for all applications or specific to a given application. The format for setting the variables is as follows:

```
cjac.app.<app key>.env.<env variable> = "<value>"|@<substitution>
```

The <app key> is the value you assign to the CJA when the CJA is called within an HTML page. You can specify any value you choose for each application. For more information about configuring CJA, see [“Setting CJA Parameters” on page 11-55.](#)

For example, to set **DBDATE** for the **stores7** application, you would include the following entry:

```
cjac.app."stores7".env.DBDATE = "DMY4/"
```

You can also create default settings for variables that will remain the same for all applications. This is done by replacing the <app key> with *. If you wanted **DBDATE** set to **DMY4** for all applications, you would specify:

```
cjac.app.*.env.DBDATE = "DMY4/"
```

Environment Variables on UNIX

The following environment variables must be set:

- **LD_LIBRARY_PATH**
- **REMOTEUSER**
(See [“Substitute Environment Variables” on page 11-40.](#))
- **INFORMIXSERVER**
- **INFORMIXDIR**
- **CLIENT_LOCALE**
- **FGLPROFILE**
- **FGLGUI**
- **FGLSERVER**
(See [“Substitute Environment Variables” on page 11-40.](#))
- **FGLLDPATH**
- **DBPATH**
- **PATH**
- **FGLDIR**

Environment Variables on Windows NT

The following environment variables must be set:

- **PATHEXT**
- **SystemRoot**
- **REMOTEUSER**
(See “Substitute Environment Variables” on page 11-40.)
- **INFORMIXDIR**
- **CLIENT_LOCALE**
- **FGLPROFILE**
- **FGLGUI**
- **FGLSERVER**
(See “Substitute Environment Variables” on page 11-40.)
- **PATH**
- **FGLDIR**

Substitute Environment Variables

The following two substitute environment variables are used for environment settings in *cjac.cnf*:

\$(FGL_GUIRVNUM)

\$(FGL_AUTHUSER)

FGLSERVER defines the Web server and port number on which CJAC is running. The **\$(FGL_GUIRVNUM)** variable captures the port number. Thus, your **FGLSERVER** entry should look like:

```
FGLSERVER=10.0.0.100:$(FGL_GUIRVNUM) or  
FGLSERVER=WebServer:$(FGL_GUIRVNUM)
```

REMOTEUSER defines the user connecting to the application. The **\$(FGL_AUTHUSER)** variable captures information about the user declared to the Web server. Thus, your **REMOTEUSER** entry should look like:

```
REMOTEUSER=" $( FGL_AUTHUSER ) "
```

Setting Commands and Arguments for Application Execution

You must also define how and where an application is to be executed. This is accomplished by creating or modifying two entries:

```
cjac.app.* | "<app key>" .cmd = "<command>"
cjac.app.* | "<app key>" .arg = "<argument>"[, ..]
```

The command entry tells CJAC how an executable is run on the local system. For UNIX, the <command> is usually `/bin/sh`; for Windows NT, it is usually `cmd.exe`.

The argument entry provides the execution instructions.

Separate command and argument entries for the same <app key> can be combined. For example:

```
cjac.app.*.cmd = "/bin/sh"
cjac.app.*.arg = "-c"
```

is equivalent to

```
cjac.app.*.cmd = "/bin/sh -c"
```

Setting General Parameters Governing CJAC Behavior

You must also specify a number of parameter settings that govern the behavior of CJAC. This section describes the following parameters:

- **cjac.comm.client.http.requestTimeout**
- **cjac.comm.client.http.requiredBandwidth**
- **cjac.comm.client.http.getTimeout**
- **cjac.comm.server.task.reannounceDelay**
- **cjac.comm.server.task.startUpTimeout**
- **cjac.comm.server.tcp.basePort**
- **cjac.comm.server.tcp.maxConnection**
- **cjac.comm.server.task.portRange**
- **cjac.comm.server.task.reuseDelay**
- **cjac.setup.check.arg**
- **cjac.setup.check.enabled**

cjac.comm.client.http.requestTimeout

Specifies, in microseconds (ms), the delay after which CJAC replies to the GUI client if the application does not reply. Specify this parameter as an integer. The default is 20000.

For example:

```
cjac.comm.client.http.requestTimeout = 20000
```

cjac.comm.client.http.requiredBandwidth

Specifies, in bytes per second, the required bandwidth for communication between the 4GL application and CJAC. Specify this parameter as an integer. The default is 500.

For example:

```
cjac.comm.client.http.requiredBandwidth = 500
```

cjac.comm.client.http.getTimeout

Specifies, in microseconds (ms), the maximum time the client waits before placing a new HTTP GET request after the last reply from the application. After this timeout, the servlet assumes that the client is not responding and shuts down the connection. Specify this parameter as an integer. The default is 30000.

For example:

```
cjac.comm.server.http.getTimeout = 30000
```

cjac.comm.server.task.reannounceDelay

Specifies, in microseconds (ms), the maximum time CJAC waits before reannouncing a restarted application to the client when the application is using RUN WITHOUT WAITING. Specify this parameter as an integer. The default is 5000.

For example:

```
cjac.comm.server.task.reannounceDelay = 5000
```

cjac.comm.server.task.startUpTimeout

Specifies, in microseconds (ms), the maximum time CJAC takes to start an application. After this time lapse, CJAC attempts to restart the application. Specify this parameter as an integer. The default is 10000.

For example:

```
cjac.comm.server.task.startUpTimeout = 10000
```

cjac.comm.server.tcp.basePort

Specifies, as an integer, the base TCP port that CJAC listens on to communicate with the 4GL application. Normally, you should not change this value; the runner uses it to communicate with CJAC. The default is 6400.

For example:

```
cjac.comm.server.tcp.basePort = 6400
```

cjac.comm.server.tcp.maxConnection

Specifies, as an integer, the maximum permissible number of simultaneous TCP connections to CJAC. The default is 10000.

For example:

```
cjac.comm.server.tcp.maxConnection = 10000
```

cjac.comm.server.tcp.portRange

Specifies, as an integer, the range of ports on which CJAC listens. For example, if the value specified for ***cjac.comm.server.tcp.basePort*** is 6400 and the value for ***cjac.comm.server.tcp.maxConnection*** is 10000, CJAC listens on ports 6400 through 16399 and allows 10000 applications to connect at the same time. The default is 10000.

For example:

```
cjac.comm.server.tcp.portRange = 10000
```

cjac.comm.server.tcp.reuseDelay

Specifies, in microseconds (ms), the maximum time CJAC waits before reusing a port for another application. Specify this parameter as an integer. The default is 20000.

For example, if user 1 runs application A on port 6600, and completes processing, and then user 2 runs application A again before the end of the reuse delay, CJAC listens to port 6601. If user 3 then runs the same application after the end of the reuse delay, CJAC listens again on port 6600.

For example:

```
cjac.comm.server.tcp.reuseDelay = 20000
```

cjac.setup.check.arg

Specifies the location of the test page. Specify this parameter as a string (usually, an HTML page). The root directory is the value of *web_server_servlet_dir* (see [“Tag Words and Paths” on page 11-9](#)).

For example:

```
cjac.setup.check.arg = "/lib/res_installation_check.html"
```

cjac.setup.check.enabled

Specifies whether troubleshooting mode is enabled. Specify this parameter as a string. The default is `true`.

For example:

```
cjac.setup.check.enabled = true
```


Sample cjac.cnf file

The following sample **cjac.cnf** file is a generic example, rather than a representation of exactly what will exist on any particular system:

```
#####
## Troubleshooting
#####

#####
# Defines whether the troubleshooting mode is enabled
# Default value : true
# Syntax :
#   cjac.setup.check.enabled = {true|false}
cjac.setup.check.enabled = true

#####
# Defines the test page
# The root directory corresponds to web_server_servlet_dir
# (see doc) Default value : "/lib/res_installation_check.html"
# Syntax :
#   cjac.setup.check.arg = "<URL>"
cjac.setup.check.arg = "/lib/res_installation_check.html"

#####
## CJCAC to Server (runner, DVM, application) communication
#####

#####
# Base TCP Port cjac listens to, in order to communicate with
# the 4GL application. DO NOT CHANGE THIS VALUE, unless a support
# engineer tells you to do it.
# Default value : 6400
# Syntax :
#   cjac.comm.server.tcp.basePort = <tcp port>
cjac.comm.server.tcp.basePort = 6400

#####
# Maximum number of simultaneous TCP connections to cjac
# Example : if *.basePort = 6400 and *.maxConnection = 10000, then
# cjac will listen from port 6400 to port 16399 and allow 10000
# applications to connect at the same time.
# Default value : 10000
# Syntax :
#   cjac.comm.server.tcp.portRange = <number>
cjac.comm.server.tcp.portRange = 10000

#####
# Delay (in ms) cjac waits before reusing this port for another
# application
# Default value : 20000
# Syntax :
```

Sample cjac.cnf file

```
# cjac.comm.server.tcp.reuseDelay = <number>
cjac.comm.server.tcp.reuseDelay = 20000

#####
## CJAC to Client (CJA, ...) communication
#####

#####
# Required bandwidth for communication between the 4GL application
# and cjac (in b/s)
# Default value : 500
# Syntax :
# cjac.comm.client.http.requiredBandwidth = <n>
cjac.comm.client.http.requiredBandwidth = 500

#####
# Maximum time (in ms) for the client to place a new GET request
# after receiving data from the servlet - after this time the
# servlet assumes the client died and shuts down the corresponding
# application
# Default value : 30000
# Syntax :
# cjac.comm.server.http.getTimeout = <number>
cjac.comm.server.http.getTimeout = 30000

#####
# After this amount of time, the client will assume that an open
# request is locked, break this request, and send a retry.
# Default value : 20000
# Syntax :
# cjac.comm.client.http.requestTimeout = <number>
cjac.comm.client.http.requestTimeout = 20000

#####
# Maximum time (in ms) for application startup. After this timeout
# occurs, CJAC will try to restart the application.
# Default value : 10000
# Syntax :
# cjac.comm.server.task.startUpTimeout = <number>
cjac.comm.server.task.startUpTimeout = 10000

#####
# When using a RUN WITHOUT WAITING, waits a maximum of time (in
# ms) before reannouncing a newly started application to the
# client.
# Default value : 5000
# Syntax :
# cjac.comm.server.task.reannounceDelay = <number>
cjac.comm.server.task.reannounceDelay = 5000
```

```
#####
## Application configuration
#####

#####
# Environment
# Default value :
# No default value is provided. If you forget to set an
# environment variable, it will not be set.
# Syntax
# cjac.app.*|"<app key>".env."<env variable>" =
# "<value>"|@<substitution>

# Example of configuration for all the applications
# cjac.app.*.env."FGLDIR"           = "/usr/fgl2c"
# cjac.app.*.env."FGLGUI"           = 1
# cjac.app.*.env."FGLPROFILE"       = "/usr/fgl2c/etc/clijava.cnf"
# cjac.app.*.env."FGLSERVER"       = "localhost:@SRVNUM"
# cjac.app.*.env."INFORMIXDIR"     = "/usr/informix"
# cjac.app.*.env."INFORMIXSERVER"  = "on_informix"
# cjac.app.*.env."LD_LIBRARY_PATH" = "/lib"
# cjac.app.*.env."PATH"            = "/usr/fgl2c/bin:/bin:/usr/bin"
# cjac.app.*.env."REMOTEUSER"      = "$(FGL_AUTHUSER)"

# Example of configuration specifically for the stores application
# cjac.app."stores".env."CLIENT_LOCALE" = "en_us.8859-1"
# cjac.app."stores".env."DBPATH"       = "/d4gldemo"
# cjac.app."stores".env."FGLLDPATH"    = "/d4gldemo"
# cjac.app."stores".env."INFORMIXSERVER" = "on_stores"

# This would be the same as writing these extra lines :
# cjac.app."stores".env."FGLDIR"       = "/usr/fgl2c"
# cjac.app."stores".env."FGLGUI"       = 1
# cjac.app."stores".env."FGLPROFILE"   = "/usr/fgl2c/etc/clijava.cnf"
# cjac.app."stores".env."FGLSERVER"    = "localhost:@SRVNUM"
# cjac.app."stores".env."INFORMIXDIR"  = "/usr/informix"
# cjac.app."stores".env."LD_LIBRARY_PATH" = "/lib"
# cjac.app."stores".env."PATH"        = "/usr/fgl2c/bin:/bin:/usr/bin"
# cjac.app."stores".env."REMOTEUSER"   = "$(FGL_AUTHUSER)"

#####
# Startup
# Default value :
# No default value is provided.
# Syntax
# cjac.app.*|"<app key>".cmd = "<command>"
# cjac.app.*|"<app key>".arg = "<argument>"[, ..]

# Example of configuration for the stores application
# cjac.app."stores".cmd = "/bin/sh"
# cjac.app."stores".arg = "-c"
# cjac.app."stores".arg = "(cd /d4gldemo; exec
#                               /usr/fgl2c/bin/fglrun d4.42r)"
```

```
# On Windows NT, these settings might be:

cjac.app.*.env."REMOTEUSER" = "$(FGL_AUTHUSER)"
cjac.app.*.env."INFORMIXDIR" = "c:\\informix"
cjac.app.*.env."CLIENT_LOCALE" = "en_us.8859-1"
cjac.app.*.env."FGLPROFILE" = "C:\\usr\\fgl2c\\etc\\clijava.cnf"
cjac.app.*.env."FGLGUI" = "1"
cjac.app.*.env."FGLSERVER" = "localhost:${FGL_GUISRVNUM}"
cjac.app.*.env."PATH" =
"C:\\WINNT;\\SYSTEM32;C:\\usr\\fgl2c\\bin;C:\\informix\\bin"
cjac.app.*.env."FGLDIR" = "C:\\usr\\fgl2c"
cjac.app.*.env."PATHEXT" = ".COM;.EXE;.BAT;.CMD;.VBS;.JS"
cjac.app.*.env."SystemRoot" = "C:\\WINNT"

cjac.app.*.cmd = "cmd.exe"
cjac.app.*.arg = "/c"
cjac.app."stores".arg = "cd /d C:\\d4gldemo
                        && C:\\usr\\fgl2c\\bin\\fglrun.exe d4.42r"

# Note that you may also create useful constants:

FGLDIR=C:\\usr\\fgl2c

cjac.app.*.env.FGLPROFILE = "$(FGLDIR)\\etc\\clijava.cnf"
```

Local and Remote Connections to the Application Server

If your Web server is connecting to the Application Server on a remote computer, you will need to make some adjustments to your *cjac.cnf* file.

When starting an application locally (with the Web server and application server on the same computer), the application server environment variables can be set directly by *cjac.cnf*.

When the Web server is connecting to the application server remotely, the environment variables specific to the application server cannot be set locally. Instead, the environment variables will usually be set by a script run on the remote application server immediately before running the application. The script and application will usually be run using a command such as **rlogin**, **rexec**, **rsh**, **telnet**, or **ssh**. The **PATH** environment variable within *cjac.cnf* allows CJAC to find the correct executable to run the remote connection mechanism.

The arguments for the remote command must be defined in the *cjac.app.*.arg* entry.

The following example shows **cjac.cnf** entries defined for remote application server access:

```
PATH="/bin"
```

(assuming **rsh** is located within **/bin**)

```
cjac.app."stores".cmd = "/bin/sh"
cjac.app."stores".arg = "-c"
cjac.app."stores".arg = "rsh AppServer -l
                        $(FGL_AUTHUSER)/d4gldemo/runwout.sh $(FGL_GUISRVNUM)"
```

The **runwout.sh** file would look something like:

```
#!/bin/sh
FGLDIR=/usr/fgl2c
export FGLDIR
PATH=$FGLDIR/bin:/bin:/usr/bin:$PATH
export PATH
FGLGUI=1
export FGLGUI
FGLSERVER=WebServer:$1
export FGLSERVER
LD_LIBRARY_PATH=/lib
export LD_LIBRARY_PATH
INFORMIXSERVER=myserver
export INFORMIXSERVER
INFORMIXDIR=/informix
export INFORMIXDIR
FGLPROFILE=$FGLDIR/etc/clijava.cnf
export FGLPROFILE
cd /d4gldemo
exec $FGLDIR/bin/fglrun d4.42r
```

Editing the *clijava.cnf* File

The **clijava.cnf** file plays a role similar to that of the file specified by your **FGLPROFILE** environment variable (the default value for **FGLPROFILE** is **fglprofile**). While the entries within this file configure the behavior of the Dynamic 4GL compiler, entries in the **clijava.cnf** file configure the behavior of the Cli Java applet.

The **clijava.cnf** file is located in the **\$FGLDIR/etc** directory on the application server. You can display the file for more information about specific entries.

The previous section (assuming the application server and Web server are on the same computer), your **cjac.cnf** file must include an entry that defines **FGLPROFILE** for each application you want to display through the Java Client. The default setting is for **clijava.cnf** because this file contains entries specific to Java applet configurations. However, you do not need to use **clijava.cnf** as your **FGLPROFILE** value. The only requirement for your **FGLPROFILE** file is that it contain the following three entries:

```
fglrun.interface = "clijava.res"  
Menu.style = 1  
gui.toolBar.enabled = 1
```

When configuring your application for display as a Java applet, it is likely that you will modify your existing **fglprofile** file as previously shown, then add entries contained within **clijava.cnf** to define how the applet will behave.

The following sections describe configuration elements within **clijava.cnf**.

Changing Colors

You can use the following entries to control the colors that appear in the foreground and background. You can also specify the exact shade of a color you want to use; for instance, if you want to use *ivory white* instead of *white*.

The following table lists the different color entries.

Background and Foreground Color Entries	Description
<code>gui.java.default.color.fg</code>	Defines the default foreground color.
<code>gui.java.default.color.bg</code>	Defines the default background color.
<code>gui.java.default.color.entry</code>	Defines the default color for entries.
<code>gui.java.default.color.entry.selected</code>	Defines the default color for selected entries.
<code>gui.java.fglcolor.fg.color</code>	Defines the RGB color used when specifying a given foreground color.

Configuring Interface Elements

You can configure different areas of the CJA interface. For instance, you can specify the behavior of the following interface elements:

- Cursor blink rate
- Fonts
- Frames
- Toolbars
- Menus

The following table lists these different interface element entries.

Interface Element Entries	Description
<code>gui.java.screen.caret.blinkRate</code>	Defines the blink rate of a caret (in ms). The caret is the cursor that appears in entry fields.
<code>gui.java.screen.workspaceFrame.font.face</code>	Defines the font used for the WorkspaceFrame.
<code>gui.java.screen.workspaceFrame.font.absoluteSize</code>	Defines the relative size of fonts used in the Java client.
<code>gui.java.screen.controlFrame.width</code>	Defines the width of the ControlFrame (in characters). The <code>gui.button.width</code> setting is ignored by the Java Client.
<code>gui.java.screen.controlFrame.button.icon.visible</code>	Defines whether each icon associated with a menu appears.
<code>gui.java.screen.toolBar.floatable</code>	Defines whether the toolbar can be moved. When the toolbar is floating, small “grips” appear on the left side.
<code>gui.java.screen.toolBar.icon.path</code>	Defines the path to the toolbar icons.

(1 of 2)

Interface Element Entries	Description
<code>gui.java.screen.menuBar.visible</code>	Defines whether the menu bar is visible.
<code>gui.java.screen.menuBar.static.help.visible</code>	Defines whether the static menuBar 'help' entry is visible.
<code>gui.java.screen.menuBar.static.help.label</code>	Defines the static menuBar entry 'help' label.
<code>gui.java.screen.menuBar.static.help.about.visible</code>	Defines whether the static menuBar 'about' entry is visible.
<code>gui.java.screen.menuBar.static.about.label</code>	Defines the static menuBar entry 'about' label.

(2 of 2)

Font Types and Known Font Equivalentents

The following list shows the supported font types and their known font equivalentents. The font types correspond to entries in the **cljjava.cnf** file. You can set the font type for fonts in the `WorkFrame` and `ControlFrame` of the CJA.

Font Type	Known Equivalentents
Dialog	Arial, Helvetica
DialogInput	Courier, Courier New
Monospaced	Courier, Courier New
SansSerif	Arial, Helvetica
Serif	Palatino, Times New Roman

Configuring Other Java Applet Elements

You can also configure the behavior of these elements of a Java applet:

- About Box
- Progress Bar

The following table lists these interface element entries.

Interface Element Entries	Description
<code>gui.java.screen.aboutBox.title</code>	Defines the title of the About box. The title appears at the top of the About box.
<code>gui.java.screen.aboutBox.label</code>	Defines the label of the About box. The label appears below the About box logo.
<code>gui.java.screen.aboutBox.logo</code>	Defines the About box logo. The logo appears in the middle of the About box.
<code>gui.java.screen.aboutBox.url</code>	Defines the About box URL. This URL is called when the user clicks the About box logo. Because of security restrictions, a Java applet can connect only to the server from which it was downloaded.
<code>gui.java.screen.progressBar.visible</code>	Defines whether the progress bar is visible.
<code>gui.java.screen.progressBar.message.send</code>	Defines the message displayed by the progress bar when sending data.
<code>gui.java.screen.progressBar.message.receive</code>	Defines the message displayed by the progress bar when receiving data.
<code>gui.java.screen.progressBar.message.refresh</code>	Defines the message displayed by the progress bar when refreshing data.

Running an Application with the Java Client

To run your application as a Java applet within a browser, you must first create an HTML page that calls the Cli Java Applet. The CJA must reside in the *web_server_clijava_dir* directory in the documents section of your Web server.

Creating the HTML Page

The HTML page you create will be the page from which users will launch the application. You will need to create a separate HTML page for each application you want to run. This page can contain anything you like, but it must include a proper call to the CJA.

The syntax for the applet call is as follows:

```
<APPLET
  CODE="com.informix.gui.applet.CJA"
  ARCHIVE="cja.jar"
  WIDTH=<width in pixels>
  HEIGHT=<height in pixels>
  CODEBASE=<path to archive>
  <PARAM NAME="AppKey" VALUE="stores">
</APPLET>
```

The **CODE** and **ARCHIVE** entries are fixed and should not be changed.

The **WIDTH** and **HEIGHT** definitions are measured in pixels and define the size of the application within the browser.

CODEBASE is used to specify the location of **cja.jar** if it is not in the current directory. In other words, if your HTML page resides in a directory other than *web_server_clijava_dir*, you will need to set **CODEBASE** to *web_server_clijava_dir*.

For example, if your HTML page resides in **web_server_dir/htdocs/clijava/stores**, you would need to set **CODEBASE** either to the directory one level above the current directory:

```
CODEBASE=".."
```

or to the absolute directory path:

```
CODEBASE="/htdocs/clijava"
```

The **PARAM NAME** and **VALUE** settings indicate a specific value for the application you want to call. This value is used by **cjac.cnf** entries to define environment variables and execution commands for specific applications. See [“Editing the cjac.cnf File” on page 11-38](#) for more information.

For more information on how **cjac.cnf** uses CJA Parameters, see [“Setting CJA Parameters” on page 11-55](#).

A sample HTML page has been created for you. This page is called **index.html** and is located in the **web_server_clijava_dir/stores** directory.

Setting CJA Parameters

You can set many parameters that define CJA behavior. Most of these parameters are included in the **clijava.cnf** file but can be defined in the HTML page calling CJA as well. The remainder can only be defined within the HTML page.

For parameters that can be set in both the HTML page and **clijava.cnf**, **clijava.cnf** settings take precedence, with the exception of **bgimage** and **bgcolor**, as explained in [“Parameter Settings not Available in clijava.cnf.”](#) If a parameter is not defined in either place, the default setting is used.

Parameter Settings not Available in clijava.cnf

You can set only the following four parameters directly in the HTML page calling CJA:

- **AppKey**. This provides a link to the **cjac.cnf** file by assigning a value to the desired application. There is no default setting. A common example of this setting is:

```
<PARAM="AppKey" VALUE="stores">
```

- **CJACPath**. This provides the path to CJAC. The default value is **/servlets/cjac**. You will not need to define this parameter unless your CJAC does not reside in this location (not recommended). An example setting is:

```
<PARAM="CJACPath" VALUE="/servlets/cjac">
```

- **bgimage.** This defines the background image of the applet and overrides the **gui.java.screen.bg.image** setting in **clijava.cnf**. There is no default setting. An example setting is:

```
<PARAM="bgimage" VALUE="/clipart/bg.gif">
```

- **bgcolor:** This defines the background color of the applet, and overrides the **gui.java.screen.bg.color** setting in **clijava.cnf**. This parameter uses the #RRGGBB syntax. The default is #FFFFFF (white). An example setting is:

```
<PARAM="bgcolor" VALUE="#FFFFFF">
```

Parameter Settings Available in clijava.cnf

Any entry in **clijava.cnf** can be defined instead within the HTML page as an applet parameter. This can be useful when you want to display the same application to different users using different parameter settings. Entries in **clijava.cnf** take precedence over applet definitions in the HTML page.

For example, if you want to define the **clijava.cnf** entry **gui.java.screen.toolBar.floatable** in the HTML page, you would add the following parameter setting to the applet definition:

```
<PARAM NAME="gui.java.screen.toolBar.floatable" VALUE="true">
```

Running the Application

Before launching your application from the browser, you should verify that the application runs properly in a character-based or Windows environment. You should also verify that your **cjac.cnf** file contains the proper entries.

To run the application, point your browser to the HTML page you created. For example, to call the **stores** application using the provided **index.html** page, you would enter:

```
http://web_server:web_server_port/web_server_clijava_dir_alias/  
stores/index.html
```

After about 30-60 seconds, you should see the basic **stores** application displayed in the browser.

The reason for the initial delay is that the CJA must be downloaded into memory the first time it is called. After the CJA is resident in memory, the application will execute more quickly. The CJA remains in memory as long as the browser remains open.

Your application should function just as it did when running in character or Windows clients, with the exceptions noted in [“Java Client Limitations” on page 11-8](#).

Java Client Enhancements

You can make the following enhancements to the Java Client interface:

- Add JavaScript to call an applet
- Use the Java Launcher to prepare Cli Java Applet startup.
- Embed the Cli Java Applet in tables, text, and so on.

See the supplementary HTML documentation included with the Java Client package for more information.

Using the Windows Client

In This Chapter	12-3
Windows Client Architecture	12-3
Windows Client Requirements	12-4
Windows 3.1 Requirements	12-4
Monitor Requirements	12-4
Dynamic 4GL Server Requirements	12-5
Remote UNIX Computer	12-5
Remote Windows NT Computer	12-5
Installing the Windows Client	12-5
After the Installation	12-6
Installing the Windows Client on a Network	12-8
Starting and Configuring the Windows Client	12-9
Starting the 4GL Server	12-9
Creating a Connection	12-9
Command-Line String Information	12-10
Connection Checking	12-11
Example	12-12
Debugging the Connection	12-13
Windows Client Language	12-13
Setting the Server Environment Variables	12-14
Using the VGA Driver with Windows 3.1	12-15
Running the Windows Client Example	12-15
Configuring the Environment Variables	12-17
Starting a P-Code Application	12-18
Authorizing the Client Computer	12-18
Starting a C-Code Application	12-19
Successful Connection	12-19

Security Features	12-20
Authorizing a Connection	12-20
Connecting Without a Password	12-21
Recording the Computer Name in the /etc/hosts.equiv File	12-21
Recording the Computer Name in the .rhosts File.	12-22
The rcp UNIX Command	12-22
Command-Line Features	12-22
Special Tags Features	12-22
ilogin Command-Line Features	12-24
Invisible Terminal Emulation.	12-26
Customizing the Login Dialog Box	12-27
Using Ataman Remote Connection Services	12-29
Adding a Scrollbar to the Terminal Emulation Window	12-30
System Colors.	12-31
Customizing the Windows Client Installation.	12-31
Customizing Icons, Titles, and Directories.	12-32
Specifying the Windows Client Icons	12-32
Installing Documentation	12-36
Configuration Files	12-37
Configuration File (WTKSRV.INI) Entries	12-37
Splash Screen Configuration	12-43
Client Configuration.	12-44
Server Configuration	12-46
Sample Configuration	12-46
User-Defined Configuration File	12-47
User-Definable WTKSRV.INI Entries	12-48
Winframe from CITRIX	12-50
First Method	12-50
Second Method	12-52

In This Chapter

This chapter describes the Windows Client installation and configuration. The Windows Client allows you to run 4GL programs in graphical mode on Windows systems.

Windows Client Architecture

The Windows Client (also known as the *WTK Client* or *Windows Front-end*) manages the interface between the client and the server. The Windows Client integrates two software components:

- **Tcl/Tk interpreter.** *Tool Command Language* (Tcl) and its *Toolkit* (Tk) are public-domain scripting languages that provide platform-independent ways of displaying graphical information and are particularly suited to GUIs and Internet applications.
- **4GL Server.** The 4GL Server was written in the Tcl language and uses the public-domain Tk3.6 port WTK from Brückner & Jarosch for MS-Windows as its interpreter. The 4GL Server runs on the client and communicates between the GUI and the 4GL commands arriving from the application server.

When executed, compiled 4GL programs generate commands for the GUI. The commands are sent to the Windows Client over a TCP/IP network to the 4GL Server, as shown in [Figure 12-1](#).

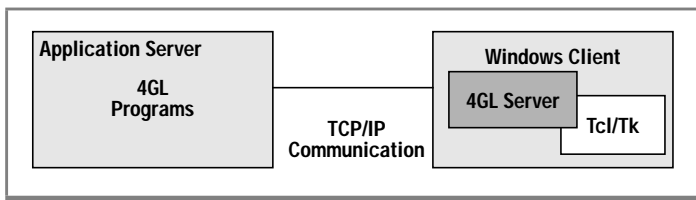


Figure 12-1
*TCP/IP
Communication of
Server Generated
Commands to the
Windows Client*



The 4GL Server listens for commands. If a graphical command is received, the command is directed to the Tcl/Tk interpreter where it is converted into graphical objects. The 4GL Server also handles UNIX remote commands (such as **rmp** or **rsh**).

Important: *The 4GL Server usually listens for commands using the socket port number 6400. This value can be modified if the port is being used by another application.*

Windows Client Requirements

Check to be sure the client system meets the following requirements:

- A Microsoft TCP/IP stack is installed and correctly configured. You must use the Microsoft TCP/IP stack. Do not use a third party TCP/IP stack.
- You can run 32-bit applications (386 or better).
- You have disabled any memory manager software (such as Emm386). Otherwise, the display speed of 4GL applications through the Windows interface will be reduced.

Windows 3.1 Requirements

For Windows 3.1, you might need to install the WIN32S extension and Microsoft TCP/IP stack. This software is included with the Dynamic 4GL media.

For directions on how to install the TCP/IP 32 stack, change to the `\WINDOWS\UTIL\TCPIP32\TCPIP32` directory on the CD and display **readme.txt**.

To install the WIN32S extension, start **setup.exe** in the `\WINDOWS\UTIL\WIN32S\DISK1` directory on the CD.

Monitor Requirements

For best results, use a screen resolution of at least 800 by 600 pixels and be able to display at least 256 colors. However, you can use the Windows Client with a standard VGA monitor.

Dynamic 4GL Server Requirements

To be able to connect to the host computer using the WTK-Rlogin option of the Windows Client, an Rlogin-Internet service must be running on the remote computer.

Remote UNIX Computer

The line **login/tcp** must be in the **/etc/services** file and a line in the **/etc/inetd.conf** file must start with the string **login**. This service is used with the rlogin program.

Remote Windows NT Computer

You can use services that allow you to simulate the rlogin feature on a UNIX computer. However, Microsoft does not provide an rlogin service for Windows NT. These services are not included with the product and must be purchased.

Dynamic 4GL provides a demonstration of an rlogin service called Ataman (see www.ataman.com). You can install and use this service for 15 days before purchasing the software. For more information on how to install Ataman, see [“Installing and Configuring the Ataman Remote Login Service” on page 2-22](#).

***Important:** You are not restricted to using the Ataman rlogin service. You can install any rlogin service for Windows NT and use it successfully with the Windows Client.*



Installing the Windows Client

The installation process installs both the WTK interpreter and the 4GL Server that make up the Windows client.

To start the installation

1. Close all applications.
2. Insert the Dynamic 4GL CD.
3. Change to the **\CLIENTS\WTK\DISK1** directory.

4. Run setup.exe.
5. Follow the directions that appear.

When prompted, install the 4GL Server. If you have an older version of the Windows client installed, you are prompted to replace the older version. If you are doing an update, the same group will be used and the files **rhosts**, **locals.tcl**, and **termuser.tcl** are not overwritten.

When prompted, specify the directory where you want to install the application. By default, the Windows client installs itself in the **\I4glsrv** directory on the partition where Windows is installed.

When prompted, enter the name of the program group to store the new icons. To use a different group name when doing an update, the previous version must be uninstalled first.



Important: Some software might be incompatible with the Windows client, such as video drivers, networks drivers, printers, spoolers, memory management programs, and Ethernet drivers. If an error occurs during the installation, try to install the Windows client with the minimum of these programs started. You can disable software in your system **autoexec.bat** or **config.sys** files.

After the Installation

After the installation, the following program icons appear in the program group specified during the installation. The default group name is Informix 4GL Server.

- The **Informix 4GL Server** icon, shown in [Figure 12-2](#), starts the 4GL Server which then runs in the background. This program listens to 4GL commands coming from the application server computer through the TCP/IP socket. The socket is defined in the Windows Client configuration file (the default value is 6400).



Figure 12-2
Informix 4GL
Server Icon

- The **Add WTK 4GL Connection** icon, shown in [Figure 12-3](#), creates new connection icons to remote hosts using the built-in terminal emulation. For information on creating connection icons, see [“Preconfiguration of Rlogin Connections”](#) on page 12-34.



Figure 12-3
*Add WTK 4GL
Connection Icon*

- The **Informix.Config.Manager** icon, shown in [Figure 12-4](#), starts the Configuration Manager for the local Microsoft Client. For more information about the Configuration Manager, see [Chapter 9, “Using the Configuration Manager.”](#)



Figure 12-4
*Informix.Config.
Manager Icon*

- The **Uninstall Informix 4GL Server** icon, shown in [Figure 12-5](#), starts the uninstallation process of the Windows client.

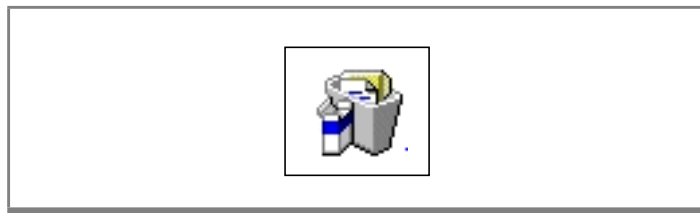


Figure 12-5
*Uninstall Informix
4GL Server Icon*

- The **Doc for WTK** icon, shown in [Figure 12-6](#), displays the online documentation. The documentation is a **.wri** file that can be displayed with the Windows text editor.

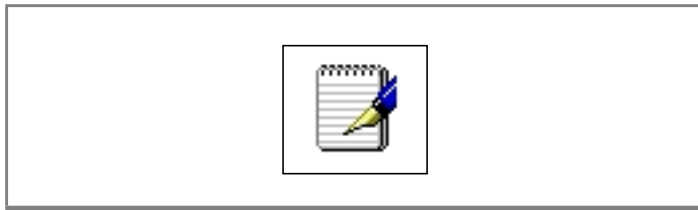


Figure 12-6
Doc for WTK Icon



Important: The Dynamic 4GL installation can be changed by editing the configuration files. For instance, you can customize the icons and help files. For more information on editing the **INSTALL.INI** configuration file, see [Chapter 9, “Using the Configuration Manager.”](#)

Installing the Windows Client on a Network

The Windows client binaries and data files can be installed from a central network server. You might want to install from a network server if you have workstations without hard disks or want to facilitate updating the client. Network file servers must be running Windows NT, Windows 95, or Windows 3.11.

For a client workstation installation, execute the application **newgrp.exe**, located in the **bin** subdirectory of the Windows Client.

For example, assume a Windows Client is installed in the **f:\fgl2cusr** directory. Computers on the network that want to install a copy of the Windows Client can mount this directory as a shared drive. They can then execute **newgrp.exe** to install the Windows Client program group.

When installing over a network, the uninstall icon is not included. To delete the Windows Client from a remote computer, you must delete the icons and the folder manually.

Starting and Configuring the Windows Client

The following sections describe how to start and configure the Windows Client.

Starting the 4GL Server

To use the Windows Client, you need to first install and start the 4GL Server. The server listens for commands in the background.

You can start the 4GL Server in either of the following two ways:

- With the 4GL Server icon in the program group created during the installation
A window appears that says that the server is started. You can copy this icon in the Windows Startup group to execute the 4GL server immediately at the beginning of each Windows session.
- With a WTK-Rlogin Telnet connection
Start a new terminal emulation with logging to a remote host; also start the 4GL Server (if needed).

Creating a Connection

The 4GL Server with TCP/IP uses the Telnet-Internet Service to create a connection.

To generate an icon for a terminal connection

1. Click the **Add WTK Connection** icon.
2. When prompted, enter the following information:
 - ❑ **Name of the computer.** The computer name is the name or the TCP/IP address of the computer that you want to connect to.
 - ❑ **Name of the user.** The name of the user is an account name existing on the remote computer.
 - ❑ **Terminal type.** The terminal type is the string that will be exported into the environment variable **TERM** of the remote computer after the connection. The **xterm** is the default.
 - ❑ **Optionally, a command-line string.** The command string might contain commands executed by the remote computer after the logging process.
3. Click **OK** to create the new connection icon.

To log onto your remote host, click the newly created icon.

The dialog box that appears can be customized with the **INSTALL.INI** configuration file. For more information, refer to [“Creating Dialog Boxes” on page 7-12.](#)

Command-Line String Information

With the optional command line added at the creation of a new connection icon, you can start a 4GL application directly by clicking an icon on the Windows client computer without need for the user to enter any UNIX commands.

The following conditions must be met to execute the optional command string that can be specified at the start of a WTK-Rlogin connection:

- The connection must be possible without a password, or the password checking feature must be enabled (see [“Connection Checking” on page 12-11.](#))
- In the **.profile** file of the HOME directory of the user on the UNIX side, there must be no interactive prompting of the user.
- The list can contain more than one command string, but they must be separated by the semicolon (;) character.

You can use the command line to enhance Windows Client login. For instance, you can automatically start the 4GL Server when initiating a Telnet connection, process command-line strings on the server after starting the connection, and customize the connection dialog boxes. These advanced features will be explained later in this chapter.

Connection Checking

During the connection phase, the 4GL Server analyzes the strings sent back by the remote UNIX computer, allowing the 4GL Server to know the state of the current connection. It knows when:

- you are prompted for the user name. It can also send your user name for you (if you have specified it).
- the UNIX server is waiting for the password, so it can let you type it or send it for you if you are using the `ilogin` (see below) feature.
- the authentication process is over; the server can then send the commands specified during the connection icon creation.

The following example shows a standard display of a terminal that is performing the authentication process:

```
fire login: F4glUser
Password:
Last login: Tue May 26 10:58:25 from fire.
```

To analyze these strings, the string should be recorded in the [RLOGIN] section of WTKSRV.INI file.

[RLOGIN] Option	Description
CHECK_PASSWORD	Set this option to 0 (zero) if you do not want to check for a password. (This is the default setting.) Set the field to 1 to enable the feature. For more information, see “User-Definable WTKSRV.INI Entries” on page 12-48.
LOGIN_QUESTION	Specifies the string sent by the UNIX server when asking for the login, usually <i>login</i> .
PASSWORD_QUESTION	Specifies the string sent by the UNIX server when asking for the password, usually <i>Password</i>
LOGIN_OK	When you successfully enter the password, the UNIX server sends a string such as "Last login: ...". Set the value to a substring of this one to tell to the 4GL Server that the authentication was successful. The next processing, such as sending the optional command line, can then be started. The LOGIN_QUESTION and PASSWORD_QUESTION strings should not be the same as for the LOGIN_OK string.

If a wrong password is typed in, the UNIX server will deny access and ask for the login prompt again.

Example

For example, suppose when making a connection to a UNIX host (for instance, a Linux computer named **fire**) the following display is sent to the terminal by the remote server:

```
fire login: F4glUser
Password:
Last login: Tue May 26 10:58:25 from fire.
Hello F4glUser
```

In this example a fourth line appears because of an entry added to the **/etc/profile** file.

Use the following settings for the connection to this computer:

- For LOGIN_QUESTION, use the string *login* that appears in the first line of the connection example. The 4GL Server then knows to automatically send the user name when the string *login* is displayed by the remote host.
- For the PASSWORD_QUESTION, use the string *word* displayed at the second line of the example. The 4GL Server knows to send a password when the string *word* is sent by the remote host.
- For the LOGIN_OK, to use the string *Last*. Use a string that appears each time the authentication process is successful. However, be careful that the strings for PASSWORD_QUESTION and LOGIN_QUESTION are not the same as the LOGIN_OK string. In this case, the second word of the third line is the word *login*. You should use the first word of this line: Last.

Debugging the Connection

To help you set the strings for the dialog between the UNIX server and the 4GL Server, you can set the field LOGIN_DEBUG to 1. A debug window displays the string comparison made between the strings set in the WTKSRV.IN file and the strings send by the remote UNIX server.

If you have more than one server, the connection strings might change from one server to another. In this case you will need to create a user-defined configuration file. For more information, see [“User-Defined Configuration File” on page 12-47](#).

Windows Client Language

By default, the language of the Windows Client is English. You can change it to German or French. To do so, you must change the value of the key language in the section [INTERNATIONAL] of the Windows Client configuration file WTKSRV.INI.

The default value is **us** for English and can be set to **de** for German or **fr** for French. This feature has nothing to do with either the language of the 4GL compiler or with the language of the 4GL application.

Setting the Server Environment Variables

After you are connected to the server, you need to specify the name or address of the client computer and the occurrence number of the communication daemon running on the Windows computer.

To do this, use the **FGLSERVER** environment variable. The syntax of the **FGLSERVER** environment variable is:

```
machine_ip_address:daemon_number
```

where the *machine_ip_address* is set to the TCP/IP address (or name) of the computer running the Windows Client and *daemon_number* is set to the occurrence number of the Windows Client.

The following example shows it in a Bourne shell:

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

These two lines tell the compiler that the GUI runs on the computer with the IP address 127.0.0.10 and uses the occurrence number 0.

In addition, you need to check the value of the **FGLGUI** environment variable. If the environment variable is set to 0, the program is executed in ASCII mode exactly as if compiled with 4GL compilers. If the environment variable is set to 1, it uses the Window Client and the application appears in graphical mode.

To check this setting, use the following UNIX command:

```
$ echo $FGLGUI
```

The following Bourne shell example shows how to set the environment variable:

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

Using the VGA Driver with Windows 3.1

The **WTK.INI** file is the main configuration file for the Tcl/Tk interpreter. The **WTK.INI** file is usually located in the **<installdir>\BIN** directory. The Tcl/Tk interpreter can use the standard VGA driver of Microsoft Windows 3.1x. To configure the driver, set the following variables.

Variables	Description
VERSION	Version of WTK
BLACKFRAMES	Set to 1 if you get black borders instead of the three-dimensional shaded frames. Useful for 16 colors configurations and border.Width set to 1.
DITHER	Set to 1 to substitute other colors (if necessary) for your special colors. This is not recommended because it can result in blotchy or mottled display for large single-color areas.

Running the Windows Client Example

This section describes the complete process for:

- configuring the Windows Client.
- creating a connection between the Windows computer named **earth** and the UNIX server named **water**.
- executing the program **ex**, as shown in [“Compiling a Simple Program” on page 3-4](#).

This example assumes that you are working on the Windows computer **earth** and have finished installing the Windows Client.

Creating a Connection

To create an **rlogin** connection between **earth** and **water**, use the WTK-Rlogin emulation.

1. Click the **Add WTK 4GL Connection** icon.

[Figure 12-7](#) shows the window that is displayed.

Figure 12-7
Connection Window
for WTK 4GL

2. Enter the following information.

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

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.

3. To validate the information, click **OK**.
You now have a new icon named **Informix@water**.
4. To start the connection, double-click the icon and enter your password when prompted.
The **rlogin** terminal is now connected to the **water** server.

Configuring the Environment Variables

If you are connecting a system, you must set the necessary environment variables, as follows:

1. Execute the shell script **envcomp** created during the installation process and located in the directory where you installed Dynamic 4GL:

```
$ . ./envfcomp
```
2. Set the **FGLSERVER** environment variable to the address of the client computer:

```
$ FGLSERVER=earth:0  
$ export FGLSERVER
```
3. Check the value of the **FGLGUI** environment variable.
If it is set to 0, the program is executed in ASCII mode exactly as if compiled with 4GL compilers. If it is set to 1, it uses the Window Client and the application appears in graphical mode.
4. Check this setting with the following command:

```
$ echo $FGLGUI
```
5. If it is set to 0 or if it is not set, set it to 1 with the following commands:

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

Starting a P-Code Application

Now that the environment variable is correctly set, start the application.

1. Change to the directory where the program is compiled.
If you have compiled it to P code, the following files are in the directory.

Type of File	Filename
The source files	ex1-1.4gl, ex1-2.4gl, ex1-1.per
The compiled form	ex1-1.42f
The P-code modules	ex1-1.42m, ex1-2.42m
The P-code link	ex1.42r

2. 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
```

Authorizing the Client Computer

Immediately after you have started the 4GL program, you will be prompted on the Windows Client asking if you want to authorize the connection. This message appears the first time a computer connects to the communication daemon running on the Client computer.

In this case, it means user **Informix** on the computer **water** is trying to access **earth**.

Select one of the following dialog box options. You have five seconds to respond to the prompt or the Client automatically responds no.

- **Yes:** allows user **Informix** on the computer **water** to have access to the local computer for all future connections. The user name and the computer name are recorded in the `\windows\rhosts` file.
- **Only once:** allows user **Informix** on the computer **water** to have access only this time.
- **No:** denies user **Informix** on the computer **water** access to the local computer.

For more information on security features, see [“Security Features” on page 12-20](#).

Starting a C-Code Application

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

Type of File	Filename
The source files	ex1-1.4gl, ex1-2.4gl, ex1-1.per
The compiled form	ex1-1.42f
The C code executable	ex1.42e

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

```
$ ex1.42e
```

Successful Connection

After a successful connection without a password, the `/etc/profile` file and the `.profile` file located in the HOME directory of the user are executed and the `TERM` environment variable is set as specified by the login icon (by default, it is set to `xterm`).

In either case, you get the same display on the computer **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, as shown in [Figure 12-8](#).



Figure 12-8
Displaying a Message Box

Security Features

Because the 4GL Server can read, write files, and execute programs using Tcl/Tk commands, the server needs to check the identity of the remote computer sending commands using the TCP port.

Authorizing a Connection

On the Windows computer running the 4GL Server, the **rhosts** file in the Windows directory lists the user/computers allowed to communicate with the 4GL server. By default, only the localhost (the Windows computer running the 4GL Server) is listed.

When a non-authorized connection to the 4GL Server occurs, a message box appears on the Windows computer and you can decide to:

- authorize the access (a new line is added to the **rhosts** file).
- authorize the access only this time.
- deny access.

To remove the access right for a previously authorized user, you must edit the **rhosts** file and remove the line specifying the user.

Connecting Without a Password

To start a connection to a computer without having to enter a password, perform the following steps:

1. Set CHECK_PASSWORD=0 in the [RLOGIN] section of the WTKSRV.INI file.

For more information, see “User-Definable WTKSRV.INI Entries” on page 12-48.

2. Update either the **host.equiv** file or **rhosts** file.

For more information, refer to the following sections.



Warning: *Setting up a password free connection creates a trusted relationship to the remote client. Anyone using the remote client can login into the server. A password free login occurs with any user name except for root.*

Recording the Computer Name in the /etc/hosts.equiv File

Record the name of the Windows computer in the **/etc/hosts.equiv** file of the UNIX computer. Record the computer name twice: once with the short computer name and again with the domain name added. You must record the name twice because one name will not be accepted, depending on whether a domain name server is used or not.

If you are connecting to a Windows NT computer, the **/etc/hosts.equiv** file is located in **\Winnt\System32\Drivers\etc\hosts.equiv**.

Recording the Computer Name in the .rhosts File

Record the name of the Windows computer in the **.rhosts** file of the **HOME** directory of the user on the UNIX computer. The **.rhosts** file must have **0600** (octal) level access rights assigned so that only the owner can read and write the file.

Using this method, an identification per UNIX user name is possible. If the Windows computer is not recorded in any of the described files, a password will be asked.

The rcp UNIX Command

The 4GL Server contains a Remote shell daemon that allows remote copy (rcp) between UNIX and MS Windows computers. The authentication works as with the 4GL Server with entries in the **rhosts** file in the Windows directory.

Command-Line Features

The following features can be added to the command line during a connection.

Special Tags Features

The following special tags can be used in the command line of the connection icons.

Tags	Description
@FGLNT	set FGLSERVER=<IP Address>&&set FGLGUI=1
@FGLCSH	setenv FGLSERVER="<IP Address>:<port>"setenv FGLGUI=1
@FGLKSH	FGLSERVER="<IP Address>:<port>";export FGLSERVER:FGLGUI=1exportFGLGUI

(1 of 2)

Tags	Description
@FGL	Replaces the "export" FGLSERVER=<ip_number>:<server_number> command with the IP number automatically set to the IP address of the client computer. You do not need any script on the UNIX server side to get the IP address of the client.
@SRVNUM	Wtk server port increment number (The second part of FGLSERVER).
@PORT	Wtk server base communication port number
@USR	Client current user name
@IP	Replaces the IP address of the client computer. This value can be used if, for example, you are using a UNIX C Shell and you cannot use the "@FGL" tag to set the FGLSERVER value.
@COMPUTER	Machine host name
E_LINES	export LINES=<Number of lines in terminal emulation window>
E_SRV	export FGLSERVER
LINES	Number of lines in terminal emulation window
4GLSRVER	Wtk server version

(2 of 2)



Important: Use the @FGL tag for backward compatibility only. For forward compatibility, use the @FGLKSH tag instead.

ilogin Command-Line Features

The **ilogin** feature allows you to customize the terminal emulation login before you start a 4GL program. To enable this functionality, change the command line in the following way:

1. Change the **login** keyword to **ilogin**, if necessary.
2. Add a flag before the user name, the computer name, the terminal type name, and the command line.

If a field flag is not defined on the command line, the corresponding field is left blank in the dialog box.

The following example shows the command line of a connection icon:

```
c:\fgl2cusr\startwtk ilogin -co myserver -cmd {  
@FGL;ia.sh;exit}
```

This command line creates a dialog box where the user enters just their user name and password.

Available flags are listed in the following table and can be abbreviated.

Flag	Value	Minimum Abbreviation
-computer	I address or I name	-co
-user	user name	-us
-termite	terminal type	-term
-candling	command line	-cant
-withdrawn	no values, make terminal invisible	-w
-visible	like -withdrawn but with value: 0 means invisible 1 means visible	-is
-all	no value, show all fields in the dialog box, and a history list box of the previous connections	-al

(1 of 3)

Flag	Value	Minimum Abbreviation
-interactive	says whether the ilogin box appears (default value=1) or the connection is immediately made without box (0)	-i
-title	specifies the text in the caption of the ilogin box	-tit
-scrabbl	specifies whether the emulation has a scrollbar(1) or not (0-default)	-scroll
-historians	specifies the number of lines in the scroll back buffer of the terminal; the buffer is created regardless of the value of scrabbl	-his
-Icon	argument must be a valid Windows .ico-File, the given Path is relative to the working directory of the 4GL Server. Example: Working dir is c:\Fgl2cusr, -Ico icons\conn1.ico, the icon must be in C:\Fgl2cusr\icons. You can also give absolute paths for the icon file name	-Ico
-height	specifies the height of the terminal window in characters (default 25)	-hei

(2 of 3)

Flag	Value	Minimum Abbreviation
-width	specifies the width of the terminal window in characters (default 80)	-wid
-autoscale	specifies whether the terminal window is scalable(1) or not(0)	-wid
-file	specifies a file where you can group common settings of a connection. With the switches in the <i>ilogin</i> command line, you can override the settings in this file. For more information, see “User-Defined Configuration File” on page 12-47.	-f

(3 of 3)

Invisible Terminal Emulation

You might want the WTK-Rlogin Terminal emulation window to disappear and appear only when required by the 4GL application (call to system functions by example) and to be lowered afterward.

To do so, edit the properties of a created icon and then add to the command line the flag **-w**.

As an example, if you have:

```
D:\usr\FGL2CUSR\BIN\STARTWTK.EXE ilogin -computer zeus -user lic -term xterm -cmd "export LINES=25"
```

change it to:

```
D:\usr\FGL2CUSR\BIN\STARTWTK.EXE ilogin -w -computer zeus -user lic -term xterm -cmd "export LINES=25"
```

or:

```
D:\usr\FGL2CUSR\BIN\STARTWTK.EXE ilogin -visible 0 -computer zeus -user lic -term xterm -cmd "export LINES=25"
```

If you want to make the Terminal emulation window appear, you just have to click **Show Wtk-Rlogin-Connections**. Select the Terminal in the list you want to be displayed and then click **switch to**.



Important: During the execution of a 4GL application, using the `fgl_system` statement instead of `run` raises the invisible WTK terminal emulation then hides it after the execution of the specified command. You have to check if the “rp” and the “hp” entries are correctly set in the UNIX termcap definition file. For more information about the `fgl_system` 4GL function, see the *INFORMIX-4GL Reference Manual*.

Customizing the Login Dialog Box

You might want to hide some of the fields of the login dialog box. To do so, add the following key to the [RLOGIN] section of the `WTKSRV.INI` file.

[RLOGIN] Key	Description
SHOW_COMPUTER	If set to 1, shows the computer frame inside the login dialog
SHOW_USER	If set to 1, shows the user frame inside the login dialog
SHOW_TERMTYPE	If set to 1, shows the terminal type frame inside the login dialog
SHOW_CMDLINE	If set to 1, shows the command-line frame in the login dialog

A few other keys can change the behavior of the login process:

(1 of 2)

[RLOGIN] Key	Description
SHOW_PROGRESS.	If the window waiting for a connection is confusing, set to 0.
KEEP_PASSWORD	If the password should be kept for the next application, set to 1. This only works if password checking is on and the server side asks for it. Keeping a password is dangerous; try to find the global variable for the kept password in remote2.tcl and if you know it, you can spy the password from the UNIX side with a simple fglUiRetrieve. It is more secure to turn it off, but for Windows NT as a server (with the Ataman remote login services), it is the only way to avoid being prompted for a password for each connection.
CANCEL_CONNECT	If set to 1, which is the default, allows cancelling a connection trial by clicking the CANCEL button of the Wtk-connect box as soon as the wait icon has disappeared. Should be set to 0 if problems occur when using non-standard TCP/IP stacks.

(2 of 2)

In addition to the CU_COMPSTR, CU_USERSTR, CU_COMPUTER, and CU_USER variables, which are also used by the Computer-User dialog, the Add-Rlogin program also uses the following variables that start with CU_:

CU_ADDTITLE	Title of the Add-Rlogin dialogs
CU_TERMSTR	Label of Terminal string
CU_DEFTERM	Default value of Terminal entry field
CU_CMDSTR	Label of optional command line
CU_DEFCMD	Default value of command entry field

The name of the icon can also be influenced when creating an Rlogin icon. There is a difference between connection with or without optional command string. For connections with the command string, the entry ADDED_TITLECMD must be used. For connections without the command string, it is ADDED_TITLE. In both entries again, %c and %u can be used as an alias for the chosen computer and user names.

Example:

```
ADDED_TITLE=Login on %c, User %u  
ADDED_TITLECMD=Command on %c, User %u
```

All variables for the Add-Rlogin program are copied in section [INSTALL] of the **WTKSRV.INI** file during installation as well as variables used in pre-configuration of rlogin.

If some variables are not set, the default settings from the standard installation are used. (To avoid unwanted settings, erase unused entries from the sample **install.ini** file or comment them out with a semicolon in the first position of the line.)

Using Ataman Remote Connection Services

Ataman allows you to create an rlogin connection between a client computer running the Windows Client and a Windows NT computer running the remote connection services.

If you are using Ataman, command syntax is the same as that used on a UNIX system except when you specify a command-line string to be executed after the connection. For this, you need to add the `\x0d\x0a` string or the command will not be executed before you hit the Return key after the connection.

The following example shows a connection icon command line starting a 4GL application after the connection to a Windows NT computer:

```
c:\fgl2cusr\startwtk ilogin -co myserver -cmd {  
c:\\usr\\fgl2c\\env.bat & set FGLSERVER=@IP:0 & cd %FGLDIR%\demo  
& fglrun72 ia & exit\x0d\x0a }
```

On Windows NT, the command separator is `&` and not `;` (as it is on UNIX).

Adding a Scrollbar to the Terminal Emulation Window

You can store the lines that scroll out from the Windows terminal into a buffer. You can then scroll backward through these lines. The following table shows the keys to use.

Keys	Description
<Shift-Up>	scrolls one line up
<Shift-Down>	scrolls one line down
<Shift-Prior>	scrolls one page up
<Shift-Next>	scrolls one page down

It is possible to add a scrollbar on the right side of the terminal to scroll through the line buffer. To enable this feature, use the following keys.

- USESCROLLBAR** If set to 1, a scrollbar on the right side of the terminal emulation allows the user to scroll backward the display of the terminal to see the history lines. If set to 0 (default value), no scrollbar is displayed.
- HISTORYLINES** Must be set to the maximum number of lines kept in the history-lines buffer. The default value is 100.

System Colors

With color management, you can use the current Windows colors settings with a 4GL application. The system color is updated each time you change the color on the system. You do not need to restart the 4GL application or Windows Client.

If the System Color Management feature is enabled, you can still use a predefined color. To do this, use the Configuration Manager to edit the **locals.tcl** file that is installed with your Windows Client software. For more information, see [Chapter 9, “Using the Configuration Manager.”](#)



***Important:** Several attributes (such as reverse and blink) are not associated with a special color on Windows. Such attributes exist only on UNIX in Text mode and for specific terminal configurations. A special color has been associated with each of these features by default.*

Customizing the Windows Client Installation

The **install.ini** file is the main configuration file for creating a customized installation. You can customize the installation in the following manner:

- Substitute different icons to be displayed.
- Rlogin configuration can be configured in advance.
- Standard components can be omitted.

The **install.ini** configuration file contains several sections in which diverse options and files can be specified. As in other Windows **.ini** files, the format of the entries is *variable=value* with only one entry per line.



***Important:** After a standard installation of the Windows Client, check the sample subdirectory for an example of the **install.ini** file using all the described possibilities.*

Customizing Icons, Titles, and Directories

You can customize the installation icons, titles of dialogs, and directory locations for installing the Windows Client. The following table lists the different keys that can be set in the **install.ini** configuration file.

Section	Description
[INTROS]	Specifies the bitmap and background color at the start of the installation
BACKGROUND	Set to 0 to remove the blue 4GL Server background
FOREGROUND	Set to 0 to remove bitmaps. The installation tool (WISE for Windows) does not allow the use of user defined bitmaps.
[TITLE]	Sets your own product name, which will appear in different dialog boxes during the installation
TITLELONG	The long name of the product
TITLESHORT	The short name of the product
[DIRECTORIES]	Allows you to change the default installation directory and the default program group name
_4GLDEFROOT	Sets the default installation directory name
_4GLDEFGROUP	Sets the default program group name

Specifying the Windows Client Icons

You can specify the icon to be added for the three standard components of the Windows Client: the 4GL Server, Add-Rlogin, and the Configuration Manager.

You can also specify which icon should be used and which name should be given to the icon. You must concatenate the `Install` section prefix and suffix to create the key.

Prefix/Suffix	Key	Description
Prefixes	SERVER ...	The 4GL Server
	ADD ...	The Add-Rlogin
	CMD ...	The Configuration Manager
Suffixes	... INSTALL	Install an icon for the prefixed component
	... TITLE	Icon name for the prefixed component
	... ICON	Icon file for the prefixed component

Example of Install Section

For instance, suppose an Add-Rlogin program has to be installed with the name *new connection* and the icon *new.ico*. You could modify the **install.ini** configuration file in the following manner:

```
[INSTALL]
...
ADDINSTALL=1
ADDTITLE=new connection
ADDICON=icons\new.ico
```

If the installation directory is **A:**, the **new.ico** file should be located in the **A:\icons** directory. However, if during the installation the **C:\account** directory is chosen as the installation directory with icon group **ACCOUNT**, then an icon *new connection* would be installed in group **ACCOUNT** with the following path and icon name: **C:\ACCOUNT\icons\new.ico**.

As shown in the previous example, a path can be specified for the icon. This path is relative to the installation directory.

Preconfiguration of Rlogin Connections

Rlogin connections can be configured in advance in the [INSTALL] section of the **INSTALL.INI** file. To do this, you must know the IP address of the 4GL application server and the programs that have to be started on the server.

NUMCONNS Specifies the number of Rlogin connection icons to create during the installation of the Windows Client.

Each connection is described using three keys. The three keys are suffixed by the number of the connection icon to create, starting with the number 0, and are prefixed by the following keys:

CONN The command line for this connection icon

CONNTITLE The icon name for this connection icon

ICONN The icon file to use for this connection icon

Example:

```
[INSTALL]
...
;2 connections
NUMCONNS=2
;first connection
CONN0=ilogin -w -co myServer -cmd { @FGL;sh holiday.sh;exit}
CONNTITLE0=Holiday application
ICONN0=icons\holi.ico
;second connection
CONN1=ilogin -w -co myServer -cmd { @FGL;sh Accounting.sh;exit}
CONNTITLE1=Accounting
ICONN1=icons\account.ico
```

In some cases the server name or the user name to use might not be known when configuring the installation. The optional Computer-User-Dialog can then be used.

If in the **INSTALL.INI** file under the `[INSTALL]` section, the key `CU_DIALOG` is set to 1, this dialog box will appear before the installation of the icons. In the description of the connection, the alias `%c` (for Computer) `%u` (or User) can be used. The dialog box can be configured with the following keys:

<code>CU_TITLE</code>	The title of the dialog box
<code>CU_COMPSTR</code>	The string display before the field where the user should enter a computer IP address or name.
<code>CU_USERSTR</code>	The string displayed before the field where the user should enter a user name.

Example:

```
;show Computer-User dialog
CU_DIALOG=1
;Title of dialog
CU_TITLE=Enter a computer and a user
;label for the computer
CU_COMPSTR=Database server
;label for User
CU_USERSTR=User:
;default value for computer entry field
CU_COMPUTER=Enter IP-Adress of server
;default value for user entry field
CU_USER=Enter user
;l conection
NUMCONNS=1
;in the comand line %c and %u will be replaced with the values of
the Computer-User dialog
CONN0=login -w %c %u vt100 "account;exit"
CONNTITLE0=Accounting on %c, User %u
ICONN0=icons\accoun.ico
;If in dialog computer name "server1" and user name "bob" have
been spcified, the icon title will be "Accounting on server1,
User bob"
```

Installing Documentation

You can install documentation, including icons that navigate to documentation. For example, you might include the files and icons for WinWrite files, text files, or WinHelp files. The NUMEXES key determines the number of icons to be installed for the documentation. An icon is determined by the following three prefixes:

EXE	Windows executable for the document
EXECMDLINE	File name of document (relative to installation medium)
EXEICON	Icon file for this document

The suffix is the order number (starting with 0).

Example:

```
NUMEXES=1
;1 document has to be installed
EXE0=write
EXECMDLINE0=doc\account.wri
;not using an icon results in the use of the standard Windows
Write icon
EXEICON0=
```

Installation of Extra Files

You can install extra files without adding icons into the program folder. To do this, you must set the NUMFILES entry to the number of files to be installed from the installation media. Then you must set the FILE(x) key, with (x) being a unique number starting at 0, to the file name to be copied.

Example:

```
NUMFILES=2
;2 files have to be installed
FILE0=bin\appl.ico
; the file bin\appl.ico have to be copied from the installation
; media into the "bin" directory of the Windows Front end
FILE1=README.lst
; the file have to be copied from the install media to the
; Windows Front-end installation media
```

Configuration Files

Because of the many options for the 4GL Server and the WTK-Rlogin terminal, a three-stage option hierarchy exists. When you set the same entry in different stages, the value is set from the last parsed configuration stage. The parse order of the three stages follows:

- There are entries in **WTKSRV.INI** file (global settings).
- There are entries in a user-defined file for grouping options, overriding entries from **WTKSRV.INI**.
- There are command-line entries for the **ilogin-Box** and **inew-Box**, overriding the settings from **WTKSRV.INI** and user-defined files.

Configuration File (WTKSRV.INI) Entries

The file **WTKSRV.INI** is the main configuration file for the 4GL Server. Typically, this file only needs to be edited to change the port number, to switch off the R shell daemon, and to configure the `[RLOGIN]` section. The file is located in the installation directory of the 4GL Server, usually **C:\FGL2CUSR**.

***Important:** Before Version 1.23, the configuration file for the 4GL Server was called **WTKSRV.INI**.*

The following table lists the most important file entries. If some entries do not appear in your file, the default values are used.

[Section]/Values	Description
[DIRECTORIES]	
WTK_ROOTDIR	Installation directory of WTK-Interpreter
WTK_EXE	WTK-Interpreter
WTK_CLIENTDIR	Directory of Tcl_Scripts for 4GL Server on the client
WTK_4GLSERVER_DIR	Same as WTK_CLIENTDIR

(1 of 7)



Configuration File (WTKSRV.INI) Entries

[Section]/Values	Description
WTK_4GLSERVER_WORKDIR	Working directory of 4GL Server
WTK_4GLSERVER_ROOTDIR	Installation directory of 4GL Server
WTK_4GLSERVER_GROUP	Program Group name of 4GL Server
[VERSION] section	
WTK_VERSION	Version number of used WTK-Interpreter
WTK_4GLSERVER_VERSION	Version number of server
[4GLSERVER]	
WTK_4GLSERVERPORT	Sets the port number of server (default=6400)
HIDE	Hides the 4GL Server during startup of Terminal and pressing Escape in the server window (default=1), to let the server minimized set to 0
ICON	Runtime Icon name for the server window (default=4glsrv.ico)
SHOW_SERVER_START	Set to 1 to display a progress box for two seconds telling the user the server started (default=1). Set to 0 (zero) to prevent the progress box from appearing.
MSG_STARTSUCCESS	Text for reporting to the user the successful start of the server
MSG_STARTFAILED	Text for reporting to the user the failure of the server-start
TITLE_SERVER	Text for the caption of the server
TITLE_CONNECTIONS	Text for the caption of the 4GL connections dialog box
[TCLVARIABLES]	

(2 of 7)

[Section]/Values	Description
WTK_USEPEERNAME	Winsock-ability (default= 1). When the application server tries to open a connection on the client machine (default port=6400) to display the 4GL application, the client first executes an authentication of the server. If WTK_USEPEERNAME is set to 1, the client tries to resolve the application server name from its numerical IP address. If authentication fails, the WTK_USEPEERNAME entry is set to 0, and the numerical IP address is recorded in the rhost file of the client machine instead of the application server name.
WTK_USEHOSTNAME	Winsock-ability (default= 1). If the 4GL Server fails to resolve the name of the computer on which it is running, The server sets the value of WTK_USEHOSTNAME to 0 and uses the numerical IP address instead of the 4GL Server name.
[INSTALL]	
ADDINSTALL	Add-Rlogin installed (default=1)
ADDTITLE	IconName Add-Rlogin
ADDICON	IconFile Add-Rlogin
SERVERINSTALL	Server-Icon installed(1)
SERVERTITLE	IconName Server
SERVERICON	IconFile Server
CFGINSTALL	4JS-Configuration Manager installed(1)
CFGTITLE	IconName Manager
CFGICON	IconFile Manager
CU_TITLE	Title for Computer-User-Dialog
CU_COMPSTR	Computer label

(3 of 7)

Configuration File (WTKSRV.INI) Entries

[Section]/Values	Description
CU_USERSTR	User label
CU_COMPUTER	Default value Computer entry
CU_USER	Default value User entry
CU_ADDTITLE	Title for Add-Rlogin-Dialog
CU_CMDSTR	Optional commands label
CU_DEFCMD	Default value command entry
CU_TERMSTR	Terminal label
CU_DEFTERM	Default value Terminal entry
ADDED_TITLE	Icon-Name of a created Rlogin-Session
DDED_TITLECMD	Icon-Name of a created Rlogin-Session with commands
[RSHD]	
ON	1 or 0 activates or deactivates RSHELL-Daemon(1)
[INTERNATIONAL]	
LANGUAGE	Two characters abbreviation of country (us, de, fr) default: us
[RLOGIN]	
sendwinsize	Default 1, if 1 sending of RFC-conform changes special sequences for Window-size (does not work with SCO-Systems)
LOGIN_COMMAND_WAIT	To set up a delay in milliseconds before sending the command line to the UNIX server. Can be used for fast client workstations.
LOGIN_DEBUG	If set to 1, a debug window will be displayed to check the dialog between the 4GL Server and the UNIX server

(4 of 7)

[Section]/Values	Description
LOGIN_QUESTION	To set up the login string sent by the UNIX server and which will be displayed in the authentication dialog box if <i>CHECK_PASSWORD</i> is enabled and the first password supplied failed
LOGIN_OK	To set up the string sent by the UNIX server once the authentication is successful
CHECK_PASSWORD	If set to 1, the password authentication process with the UNIX server will be enabled
PASSWORD_QUESTION	To set up the password string sent by the UNIX server and which will be displayed in the authentication dialog box if <i>CHECK_PASSWORD</i> is enabled
SHOW_COMPUTER	If set to 1, shows the computer frame inside the login dialog
SHOW_USER	If set to 1, shows the user frame inside the login dialog
SHOW_TERMTYPE	If set to 1, shows the terminal type frame inside the login dialog
SHOW_CMDLINE	If set to 1, shows the command-line frame in the login dialog
KEEP_PASSWORD	<p>If the password should be kept for the next application, set to 1. This only works if password checking is on and the server side asks for it.</p> <p>Keeping a password is dangerous; try to find the global variable for the kept password in <i>remote2.tcl</i> and if you know it, you can spy the password from the UNIX side with a simple <i>fglUiRetrieve</i>. It is more secure to turn it off, but for Windows NT as a server (with the Ataman remote login services), it is the only way to avoid being prompted for a password for each connection.</p>

(5 of 7)

[Section]/Values	Description
SHOW_PROGRESS.	If the window waiting for a connection is confusing, set to 0
CANCEL_CONNECT	If set to 1, which is the default, allows to cancel a connection trial by clicking on the CANCEL button of the Wtk-connect box as soon as the egg timer has disappeared. Should be set to 0 if trouble occurs when using nonstandard TCP/IP stacks.
USESCROLLBAR	If set to 1, a scrollbar on the right side of the terminal emulation allow the user to scroll backward the display of the terminal to see the history lines. If set to 0 (default value), no scrollbar is displayed.
HISTORYLINES	Must be set to the maximum number of lines keep in the history lines buffer. The default value is 100.
ILOGIN_INTERACTIVE	Tells the ilogin dialog if it should run with a box(default:1) or immediately(0)
ILOGIN_TITLE	Title in the caption of the ilogin-dialog
ILOGIN_TITLE_FAILED	Title of the ilogin-box if the login fails
ILOGIN_TITLE_PASSWORD	Title of the ilogin-box when expecting a password
ILOGIN_TITLE_CREATEICON	Title of the inew-box when creating an icon
ILOGIN_ENTRYWIDTH	Width of the entries in the ilogin-dialog (in characters)
ILOGIN_TXT_COMPUTER	Label text for the computer (replaces the former CU_COMPSTR in [INSTALL])
ILOGIN_TXT_USER	Label text for the user (replaces the former CU_USERSTR in [INSTALL])
ILOGIN_TXT_TERMTYPE	Label text for the user (replaces the former CU_TERMSTR in [INSTALL])

[Section]/Values	Description
ILOGIN_TXT_CMDLINE	Label text for the user (replaces the former CU_CMDSTR in [INSTALL])
COMPUTER	Default in the computer entry (replaces the former CU_COMPUTER in [INSTALL])
USER	Default in the user entry (replaces the former CU_USER in [INSTALL])
TERMINAL	Default in the terminal entry (replaces the former CU_DEFTERM in [INSTALL])
OPTCOMMAND	Default in the command-line entry (replaces the former CU_DEFCMD in [INSTALL])
ILOGIN_TXT_PASSWORD	Label text for the password
ILOGIN_TXT_VISIBLE	Label text for the visible box
ILOGIN_TXT_INTERACTIVE	Label text for the interactive box

(7 of 7)

Splash Screen Configuration

You can create or modify a splash screen. You can configure the splash screen on the client and server.



Important: You must install Version 3.0 of the Windows Client to use the splash screen feature.

Client Configuration

To configure the client, change the values in the **Wtksrv.ini** file. A new [SPLASH] resource has been added to the file. The following table lists the basic configuration resources.

[Section]/Values	Description
[SPLASH]	
SPLASH_VISIBLE	Disable or enable a splash screen: 0: No splash screen appears 1: A splash appears. The default is 1.
SPLASH_BACKGROUND	Specifies the background color for the splash frame. A color name or color hex value can be used. The default value is the current system background color.
SPLASH_FOREGROUND	Specifies the foreground color for the splash frame. A color name or color hex value can be used. The default value is the current system background color.
SPLASH_DURATION	Specifies the duration of the splash screen display in seconds. The default value is 1.
SPLASH_LAYOUT	Select from four different splash screen layouts: 1: Display bitmap and text on the left side 2: Display bitmap and text on the right side 3: Display bitmap and text on top 4: Display bitmap and text on bottom The default value is 0.

The following table lists the different bitmap splash screen configurations.

[Section]/Values	Description
[SPLASH]	
SPLASH_BITMAP_USE	Specifies if a bitmap appears in the splash screen. 1: Splash bitmap enable. 0: Splash bitmap disable. The default value is 1.
SPLASH_BITMAP_FILE	Specifies the path to the bitmap file. The path you should use '/' or '\\\' as directory separators. For example, SPLASH_BITMAP_FILE="C:\\WINNT\\256.bmp"
SPLASH_BITMAP_COLOR_FILTER	Applies a color filter to colorize the bitmap file. The color can be the name or the hex value. The default value is #ffff.

The following table lists the different text splash configurations.

[Section]/Values	Description
[SPLASH]	
SPLASH_TEXT_STRING	Specifies the display text. The default value is blank.
SPLASH_TEXT_FONT_NAME	Specifies the font name to use for the display text. The default value is Arial.
SPLASH_TEXT_FONT_STYLE	Specifies the font style to use. For example, you can use bold, normal, or italic. The default value is normal.

(1 of 2)

[Section]/Values	Description
SPLASH_TEXT_FONT_SIZE	Specifies the font size. The default size is 12 points.
SPLASH_TEXT_FOREGROUND	Specifies the foreground color text. A color name or color hex value can be used. The default value is the current system background color.
SPLASH_TEXT_BACKGROUND	Specifies the background color text. A color name or color hex value can be used. The default value is the current system background color.

(2 of 2)

Server Configuration

To configure the server, you can edit values in the **fglprofile** file. The values will appear in the following format:

```
splash.<Application name>.<type>
```

The following example shows the same value in the **wtkserv.ini** and **fglprofile** files:

```
wtksrv.ini      SPLASH_BITMAP_FILE=e:/bmpfiles/ia.bmp
fglprofile      splash.ia.bitmap.file="e:/bmpfiles/ia.bmp"
```

Sample Configuration

The following example shows a configuration file for the client (with a fast cpu speed) and server. The configuration file on the client specifies information about the splash screen; the one on the server species the content.

```
[SPLASH]
SPLASH_VISIBLE=1
SPLASH_BITMAP_USE=1
SPLASH_BACKGROUND="red"
SPLASH_FOREGROUND="black"
SPLASH_DURATION=5
SPLASH_LAYOUT=2

splash.ia.bitmap.file="e:/bmpfiles/ia.bmp"
splash.ia.bitmap.color.filter="white"
splash.ia.text.string="Informix Tools"
splash.ia.text.font.name="Courier"
splash.ia.text.size="12"
splash.ia.text.foreground="black"
splash.ia.text.background="red"
```



Important: The **SPLASH_BITMAP_USE** resource is only available on the client side.

User-Defined Configuration File

You can create a small **Tcl/Tk** script file to set the variables that correspond to the keys of the **WTKSRV.INI** file. You add a **-f** flag to the command line of the connection icon followed by your user-defined filename.

The following example shows a modified command line:

```
<windows front end>\BIN\STARTWTK.EXE ilogin -comp {water} -us  
{4js} -cmdline {@FGL} -f c:/user/4js/wtk.cnf
```

The path is given with / character and not \.

The next step is to write the **c:\user\4js\wtk.cnf** user-defined configuration file. Of course you can choose the name you want for these configuration files. This file is a simple **Tcl/Tk** script setting the **Tcl/Tk** variables listed in [Appendix A](#).

The **Tcl/Tk** syntax for setting variables is as follows:

- One command per line
- The name of the variables are case sensitive
- The syntax is: `set <variable name> <value>`
- If the `<value>` is a string of more than one word, enclose the string between double quotes

The following example shows a **Tcl/Tk** script:

```
set termttype xterm  
set user 4js  
set cmdline "@FGL;cd /usr/4js;/fglrun apps;exit"  
set login_ok "Successful"
```

User-Definable WTKSRV.INI Entries

Entries for user-defined configuration files and command-line options are shown in the following table.

WTKSRV.INI Entries	User-Defined Configuration Files	Command-Line Options
[RLOGIN] section		
sendwinsize	sendwinsize	sendwinsize
LOGIN_COMMAND_WAIT	login_command_wait	noentry
LOGIN_DEBUG	noentry	noentry
LOGIN_QUESTION	login_question	noentry
LOGIN_OK	login_ok	noentry
CHECK_PASSWORD	check_password	noentry
PASSWORD_QUESTION	password_question	noentry
SHOW_COMPUTER	show_computer	noentry
SHOW_USER	show_user	noentry
SHOW_TERMTYPE	show_termtype	noentry
SHOW_CMDLINE	show_cmdline	noentry
SHOW_VISIBLE	show_visible	noentry
SHOW_INTERACTIVE	show_visible	noentry
KEEP_PASSWORD	noentry	noentry
SHOW_PROGRESS.	show_progress	noentry
CANCEL_CONNECT	noentry	noentry
COMPUTER	computer	computer
USER	user	user
TERMINAL	termtype	termtype

(1 of 2)

WTKSRV.INI Entries	User-Defined Configuration Files	Command-Line Options
OPTCOMMAND	cmdline	cmdline
VISIBLE	visible	visible
USESCROLLBAR	usescrollbar	scrollbar
HISTORYLINES	historylines	historylines
WIDTH	width	width
HEIGHT	height	height
AUTOSCALE	autoscale	autoscale
ICON	icon	Icon
ILOGIN_INTERACTIVE	interactive	interactive
ILOGIN_TITLE	ilogin_title	title
ILOGIN_ENTRYWIDTH	entrywidth	noentry
ILOGIN_TITLE_FAILED	ilogin_title_failed	noentry
ILOGIN_TITLE_FAILED	ilogin_title_failed	noentry
ILOGIN_TXT_USER	txt_user	noentry
ILOGIN_TXT_PASSWORD	txt_password	noentry
ILOGIN_TXT_CMDLINE	txt_cmdline	noentry
ILOGIN_TXT_VISIBLE	txt_visible	noentry
ILOGIN_TXT_INTERACTIVE	txt_interactive	noentry
NAME_PROGMAN	name_progman	noentry
RLOGIN_TITLE	noentry	noentry
DIALOG_ICON	noentry	noentry

(2 of 2)

Winframe from CITRIX

Winframe is a multiuser port of Windows NT, Version 3.51, from Citrix. Thus, through a small client (the ICA client), any user can use applications compliant with Windows NT, Version 3.51, such as the Windows Client. This makes maintenances much easier because everything is stored and running on a unique computer.

However, the Windows Client needs to be configured to avoid the problem of conflict between simultaneous users of each Windows Client. The problem is that each occurrence of the Windows Client needs its own socket port to communicate.

Two possible solutions are to:

- install a different version of the WTK for each user and then set the entry `WTK_4GLSERVERPORT` in the `WTKSRV.INI` configuration file to a different value. This option is disk-space intensive and might be time consuming.
- install a single copy of the Windows Front-end and use one of the two methods shown in [“First Method” on page 12-50](#) and [“Second Method” on page 12-52](#).

First Method

With this method, the 4GL runtime package will try to automatically start the client graphical daemon when a 4GL application is started. This is possible because the 4GL application is running on the same computer as the one running the graphical daemon. If you are using the Winframe computer only for running the Windows front-end but the 4GL applications are running on another computer (Windows NT or UNIX), you have to use the second method.

Winframe sets the `WINSTATIONNAME` environment variable. This environment variable follows the following structure:

```
aaaaaa#nnnnn
```

where:

- `aaaaaa` is the connection type (WinCenter, tcp, and so forth)
- `nnnnn` is a unique connection type number

One known exception for the **WINSTATIONNAME** environment variable is that it can be set to **Console** when you are working directly on the Winframe console.

In this example, suppose the Winframe server is named **MYSERVER**.

When starting a 4GL application (for example, `fglrun ia.42r`), the following events might occur:

- If **FGLSERVER** is set, the application will use it.
- If **FGLSERVER** is not set, you must unset the **DISPLAY** environment variable (otherwise, the F4GL runner assumes you are using a UNIX X11 server, which is not the case).
- Extract the **nnnn** from the **WINSTATIONNAME** environment variable and add 1 and this becomes the server number.

Example:

```
WINSTATIONNAME="WinCenter#002" The server number will be 3.
WINSTATIONNAME="Console" The server number will be 0 as the only
known exception is "Console".
```

At this stage, the runner tries to connect to **MYSERVER:nnnn**. If it does not succeed, the runner will launch:

```
fglssrv -n nnnn
example:
fglssrv -n 2
```

This command can be set by **fglrun.server.cmd** in the **fglprofile** file. Then the runner tries to reconnect to **FGLSERVER=MYSERVER:nnnn**. If it still does not succeed, the runner says: error -1400.

The **fglssrv** command is a batch file located in the **\$FGLDIR/bin** directory that starts the 4GL Server with the port number passed as an argument incremented by 6400.

Sometimes it might happen that two **WINSTATIONNAME** environment variables are using the same **nnnn** number with two different connection types. This can only occur when connections to the WINFRAME client are coming from different kinds of clients. If you have only one kind of connection client (ICA, Wincenter connect from NCD), this problem should not occur.

Second Method

This method allows you to launch the 4GL graphical server with a unique port number when the computer starts up.

You should put in your Startup group a copy of the WTK 4GL Server icon and add at the end of the target line the word *AUTO*. The group name depends on the language version you are using. For example:

Target:

```
C:\usr\FGL2CUSR\BIN\WTK.EXE -f C:\usr\FGL2CUSR\FGL2C\doserv1.tcl  
AUTO
```

Start in:

```
c:\usr\fgl2cusr
```

Where, of course, **c:\usr\fgl2cusr** is the installation directory for WTK.

In this case, when the user logs into your Winframe server, a graphical daemon is started with a unique number. With the tags like the @FGL strings, you will be able to send the correct port number to the 4GL application server with the command-line option of the *ilogin* connection feature.

The main default of this method is that the 4GL Server is always running even if you do not start a 4GL application during the Winframe session.

Using the X11 Client

In This Chapter	13-3
UNIX X11 Client Configuration	13-3
Installing the X11 Client	13-4
Prerequisites	13-4
Installing Tcl/Tk	13-4
Manually Installing Tcl/Tk	13-5
Installing the X11 Daemon	13-5
Setting the Tcl/Tk Environment Variables	13-5
Managing Application Windowing	13-5
Running the Program on the X11 Client	13-8
Displaying the TCL Interpreter	13-8
Configuring the Environment	13-10
Starting the Application Using the X11 Client	13-11

In This Chapter

This chapter describes the X11 interface, including how to install and configure the software.

UNIX X11 Client Configuration

The X11 Client displays your 4GL applications in graphical mode in X11-compliant interfaces. The X11 Client is made up of the following two software components:

- Tcl/Tk interpreter
- X11 daemon that manages communication between the Tcl/Tk interpreter and the 4GL runner.

All communication uses the TCP/IP protocol, which allows the components to be installed on different computers, as shown in [Figure 13-1](#).

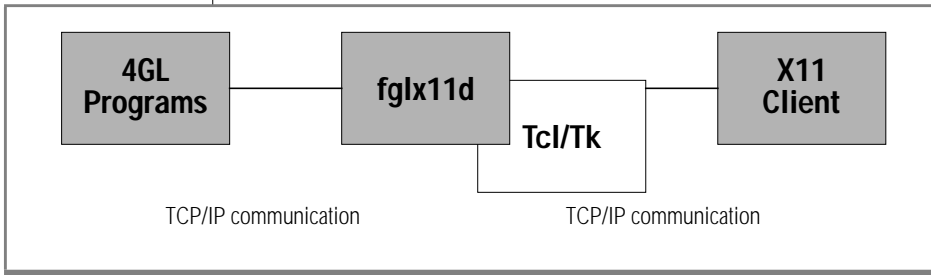


Figure 13-1
*TCP/IP
Communication
Enables
Components to be
Installed on
Different
Computers*

Installing the X11 Client

The Tcl/Tk interpreter is included with a separate installation on the Dynamic 4GL CD. You must use this enhanced version of the Tcl/Tk interpreter. The X11 daemon called **fglX11d** is installed with both the UNIX Dynamic 4GL development package and the runtime package.

Prerequisites

The X11 interface displays best using a monitor with at least 256 colors.

Installing Tcl/Tk

You must install Tcl/Tk first. You do not need to run this shell as **root**. However, you need to have sufficient permissions to create the directories where you want to install the Tcl/Tk package.

Follow these steps to install Tcl/Tk:

1. Mount the Dynamic 4GL CD on your file system:

```
$ mount your_cdrom_device_name /cdrom
```

Depending on your system, the syntax of the mount command can be different. Check your UNIX manual. Also, depending on your system, the names of the files located on the CD might be in either lowercase or uppercase letters.

2. Go to the **/cdrom/OS/UNIX** directory.
3. Enter the following command to start the installation process:

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

If you do not specify **-i** flag, you get the syntax help message.

The installation determines your operating-system name and checks for a few requirements, including if you already have a Tcl/Tk package installed.

A prompt appears for the installation directory and starts copying the files 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 sets the needed environment variables to make the Tcl/Tk interpreter work. This script is written in a Bourne shell.

Manually Installing Tcl/Tk

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

Follow the installation directions described in step 3 in [“Running the Program on the X11 Client” on page 13-8](#) to run the installation shell.

Installing the X11 Daemon

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

Setting the Tcl/Tk Environment Variables

The two environment variables that the **envtcl** file sets are:

- **TCL_LIBRARY**—the path to the **tcl** libraries
- **TK_LIBRARY**—the path to the **tk** libraries

The **envtcl** file also adds the **/bin** subdirectory to the **PATH** environment variable.

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**).

Managing Application Windowing

The **fglX11d** daemon manages application windowing. One occurrence of this daemon must be started for each X11 display.

This daemon can run on a computer other than the one where the 4GL program runs or the one where the output is displayed.

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, as shown in [Figure 13-2](#).

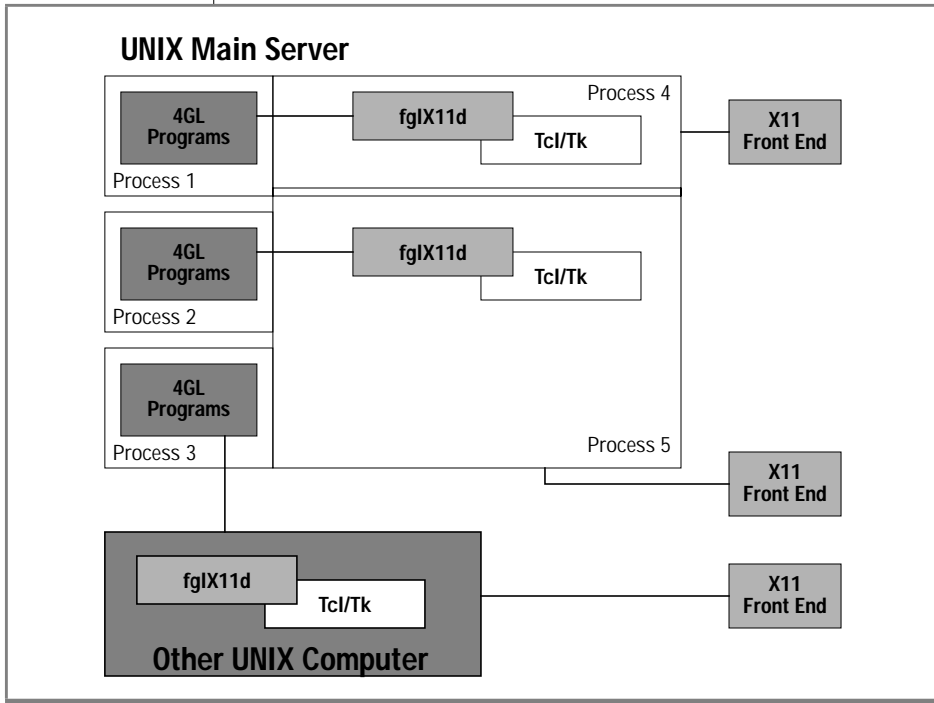


Figure 13-2
Application
Windowing
Management
Architecture

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** environment variable is:

machine_ip_address:daemon_number

where *machine_ip_address* is set to the TCP/IP address or name of the computer running the **fglX11d** daemon, and *daemon_number* is set to the occurrence number of the **fglX11d** daemon. Each daemon started on one computer should have a unique occurrence number.

Here is an example in a Bourne shell:

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

These two lines tell the compiler the graphical daemon will run on the computer with the IP address 127.0.0.4 and use the daemon number 9.

The second needed environment variable is **DISPLAY**, which tells the **fglX11d** daemon which X11 server it must use for the graphical output (to your client computer). This environment 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 computer, 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 computer 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 with the following syntax:

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

The following table describes the options in this command.

Option	Description
-n <i>daemonNumber</i>	Single ID to identify multiple daemon occurrences on one host (default: 0)
-w <i>wishName</i>	Name of the visual shell to be used (default: wish)
-f <i>scriptName</i>	Name of the script to be used to initialize the server (default: \$FGLDIR/etc/fgl2c.tcl)
-s <i>portNumber</i>	TCP port to be used (default: 6400)
-e <i>daemonNumber</i>	Occurrence number of the daemon to shut down (default: 0)

(1 of 2)

Option	Description
-l	Logs all traffic to stderr
-v	Gives the version information and exits
-a	Gives the number of the next free daemon

(2 of 2)

Because this is a daemon, 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.

Running the Program on the X11 Client

This section describes how to configure the X11 Client (UNIX or Windows) to run a sample compiled program.

For this example, the compiler and 4GL programs are on a UNIX server with a TCP/IP name set to **water**. The program will be displayed using the X11 interface on a client computer named **fire**. The client computer **fire** has a **telnet** or **rlogin** connection to the **water** computer.

Displaying the TCL Interpreter

1. Set the Tcl/Tk environment variables using the **envtcl** shell script.
If Tcl/Tk is installed in the **/usr/local/tcltk** directory, then the **envtcl** file is located in this directory:

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

2. Check that the environment is correctly configured with the following two commands:

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

3. Set the **DISPLAY** environment variable with the following command so that the computer **water** sends all graphical output to the computer **fire**:

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

All graphical programs started in this environment will now be displayed by the X11 server number 0 of the computer **fire**.

4. Start the Tcl/Tk interpreter to check if it is correctly installed and if the **DISPLAY** environment variable is correctly set:

```
$ 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, as [Figure 13-3](#) shows.

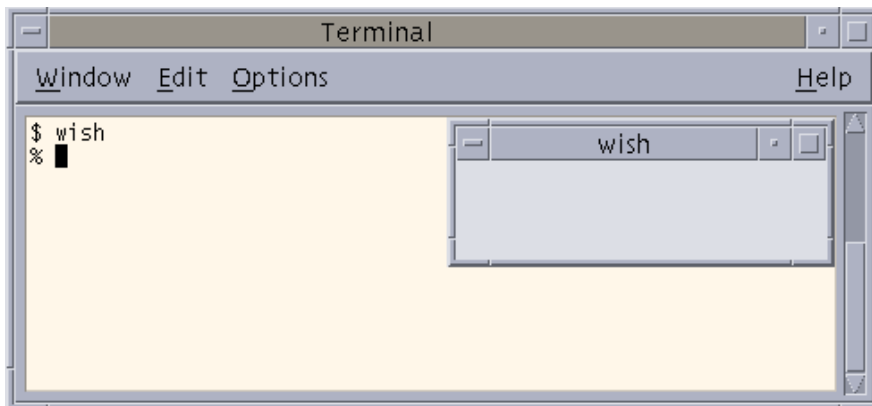


Figure 13-3
Tcl/Tk
Interpreter

5. To quit, enter:

```
% exit
```

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

Configuring the Environment

Typically, the first user to run a graphical 4GL program on the computer **water** would use the number 0 as the occurrence number for the **fglX11d** daemon.

1. For this example, use 5 as the occurrence number:

```
$ fglX11d -n 5 &
```

Add the ampersand character (&) to the command line to cause the daemon to execute in the background. The prompt can then be available for new commands.

2. Add the **FGLSERVER** environment variable to tell the 4GL program which daemon to use.

In this case, the daemon is running on the computer **water** and the occurrence number is 5:

```
$ FGLSERVER=water:5  
$ export FGLSERVER
```

3. Check the value of the **FGLGUI** environment variable with the following command:

```
$ echo $FGLGUI
```

If it is set to 0, the program will be executed in ASCII mode (exactly as if compiled with 4GL compilers). If set to 1, it will use the **fglX11d** daemon and the application appears in graphical mode.

4. Set the **FGLGUI** environment variable to 1 with the following commands:

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

Starting the Application Using the X11 Client

Now that you have set the environment variable, start the application.

1. Change to the directory where the program is compiled.

If you have compiled it to P code, the following files are in the directory.

Type of File	Filename
The source files	ex1-1.4gl, ex1-2.4gl, ex1-1.per
The compiled form	ex1-1.42f
The P-code modules	ex1-1.42m, ex1-2.42m
The P-code link	ex1.42r

2. 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.

In this example, the runner can be **fglnodb** because 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.

Type of File	Filename
The source files	ex1-1.4gl, ex1-2.4gl, ex1-1.per
The compiled form	ex1-1.42f
The C-code executable	ex1.42e

In this case, run the C-code executable:

```
$ ex1.42e
```

In either case, you get the same display on the computer **fire**.

3. Click the **Message box** menu item.
4. Select one of the four icon radio buttons.
5. Enter a title and a message into the appropriate text boxes.

Environment Variables

This appendix provides a complete list of all environment variables for use with Dynamic 4GL.

Some environment variables are only available on UNIX systems. The description section states whether the environment variable is only available on UNIX.

Dynamic 4GL Environment Variables

This appendix provides a brief description of each environment variable and the possible values you can set for it, with examples of how to set the environment variable on the available platforms.

FGLGUI

Description	This environment variable determines if the Dynamic 4GL programs will run with a character-based ASCII user interface or a graphical user interface (GUI).	
Values	0 or not set	The 4GL application executes using ASCII mode
	1	The 4GL application executes using graphical mode
Default	0 on UNIX 1 on Windows	
Korn shell	<code>\$ export FGLGUI=1</code>	Korn shell
C shell	<code>\$ setenv FGLGUI 1</code>	C shell
Microsoft DOS	<code>C:\> set FGLGUI=1</code>	Microsoft DOS

FGLDBPATH

Description	This environment 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 .	
Values	The path to the schema file	
Default	Set to the current directory	
Korn shell	<code>\$ export FGLDBPATH=/schema:\$FGLDBPATH</code>	
C shell	<code>\$ setenv FGLDBPATH "/schema:\$FGLDBPATH"</code>	
Microsoft DOS	<code>C:\> set FGLDBPATH=C:\schema:%FGLDBPATH%</code>	

FGLDIR

Description	This environment variable contains the path to the installation directory. This environment variable is required when you use either the development package or the runtime package of Dynamic 4GL.
Values	The path to Dynamic 4GL
Default	<p>UNIX: /usr/fgl2c</p> <p>Windows: C:\usr\fgl2c</p>
Korn shell	<code>\$ export FGLDIR=/usr/fgl2c</code>
C shell	<code>\$ setenv FGLDIR "/usr/fgl2c"</code>
Microsoft DOS	<code>C:\> set FGLDIR=C:\usr\fgl2c</code>

PATH

Description	This system variable contains the path to the binary programs. Add the path to the Dynamic 4GL binary program.
Values	The path to the binary directory
Korn shell	<code>\$ export PATH=\$FGLDIR/bin:\$PATH</code>
C shell	<code>\$ setenv PATH \$FGLDIR/bin:\$PATH</code>
Microsoft DOS	<code>C:\> set PATH=%FGLDIR%\bin;%PATH%</code>

FGLCC

Description	Available only on UNIX. This environment variable must be set when you want to compile a new runner.
Value	The name of the C or C++ compiler
Korn shell	<code>\$ export FGLCC=gcc</code>
C shell	<code>\$ setenv FGLCC gcc</code>

FGLRUN

Description	This environment variable must be set to the name of the specific P-code runner when linking P-code modules using fgl2p -o . Use this environment variable for modules calling C functions that have been linked to this runner by the fglmkrun utility.
Value	The name of the runner that you currently use
Default	<code>FGLRUN=fglrun</code>
Korn shell	<code>\$ export FGLRUN=fglrun</code>
C shell	<code>\$ setenv FGLRUN fglrun</code>
Microsoft DOS	<code>C:\> set FGLRUN=fglrun</code>

FGLLDPATH

Description	This environment 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	<code>\$ export FGLLDPATH=/modules:\$FGLLDPATH</code>
C shell	<code>\$ setenv FGLLDPATH /modules:\$FGLLDPATH</code>
Microsoft DOS	<code>C:\> set FGLLDPATH=c:\modules:%FGLLDPATH%</code>

FGLLIBSQL

Description	Available only on UNIX. This environment variable specifies the complete path to the SQL library, to link with the P-code runner or the C-code programs that contain the interface functions to the database server.
Value	Complete path to the SQL library
Default	<code>\$INFORMIXDIR/lib/libfesql.a</code>
Korn shell	<code>\$ export FGLLIBSQL=\$INFORMIXDIR/lib/libfesql.a</code>
C shell	<code>\$ setenv FGLLIBSQL \$INFORMIXDIR/lib/libfesql.a</code>

FGLLIBSYS

Description	Available only on UNIX. This environment 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	<code>\$ export FGLLIBSYS="-lm -lsocket"</code>
C shell	<code>\$ setenv FGLLIBSYS "-lm -lsocket"</code>

FGLSQLDEBUG

Description	If set to 1, this environment 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 environment 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	<code>\$ export FGLDEBUGON=0</code>
C shell	<code>\$ setenv FGLDEBUGON 0</code>

GCC Environment Variables

These environment variables are only available on UNIX.

CC

Description	Available only on UNIX. This environment 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	<code>\$ export CC="cc"</code>
C shell	<code>\$ setenv CC cc</code>

GCC

Description Available only on UNIX. This environment variable specifies the name of the GNU C Compiler.

Value The name of the GNU C compiler.

Korn shell `$ export GCC=gcc`

C shell `$ setenv GCC gcc`

GCCDIR

Description Available only on UNIX. This environment variable specifies the directory in which the GNU C compiler is installed. This environment variable is used only by Dynamic 4GL

Value The path of the **gcc** installation directory

Korn shell `$ export GCCDIR=/usr/local/gcc-2.80`

C shell `$ setenv GCCDIR /usr/local/gcc-2.80`

GCC_EXEC_PREFIX

Description Available only on UNIX. This environment variable specifies the path of the installation directory of the GCC compiler.

Value Path to the **gcc** installation directory

Korn shell `$ export GCC_EXEC_PREFIX=/usr/local/gcc-2.80`

C shell `$ setenv GCC_EXEC_PREFIX /usr/local/gcc-2.80`

PATH

Description	Available only on UNIX. This environment 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	<code>\$ export PATH=\$GCCDIR/bin:\$PATH</code>
C shell	<code>\$ setenv PATH \$GCCDIR/bin:\$PATH</code>

Tcl/Tk Environment Variables

These environment variables are available only on UNIX.

TCLDIR

Description	Available only on UNIX. This environment variable is used only with the Tcl/Tk package included in Dynamic 4GL. This environment variable specifies the full path to the installation directory of the Tcl/Tk .
Value	Complete path to the Tcl/Tk installation directory
Korn shell	<code>\$ export TCLDIR=/usr/local</code>
C shell	<code>\$ setenv TCLDIR /usr/local</code>

TK_LIBRARY

Description Available only on UNIX. This environment 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 environment 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 environment 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`

Common Problems and Workarounds

This appendix contains information about how to resolve issues in the following areas:

- Installing the Dynamic 4GL software manually
- The interruption signal
- The P-code runner and C-code compilation
- Special characters and the GLS feature
- The Windows Client
- 4GL program errors
- X11 issues
- Windows issues

Installing the Dynamic 4GL Software Manually

If you have problems installing the Dynamic 4GL software, you can perform a manual installation.

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 uncompressing 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 following 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:

```
$ 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:

```
$ 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:

```
$ 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:

```
$ 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:

```
$ cd $FGLDIR/bin
$ licencef4gl
```

This will start the license installation process, as described in [“Licensing the Software” on page 2-11](#).

Post-Installation Tasks

If you are doing a manual installation, you need to complete the following procedures by hand before you can 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 computer or the C compiler of the GNU tools, included on the distribution media.

If there is no usable native C compiler on your computer, the GNU tools must be installed. However, 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 script installs the package **GCC.TGZ** located in the directory **/OS/UNIX/your_OS_name/GNUC**.

If you cannot mount the CD directly on UNIX, you can copy the file **gnuc.sh**, located in the directory **/OS/UNIX/your_OS_name/SELFEXTR**, to a temporary directory on UNIX. 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 script to set all the environment variables needed for compiling and linking C programs.



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 Dynamic 4GL 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 **envcomp** 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:

```
$ ./envcomp
```

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” on page 4-8](#).

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



To build a P-code runner, type:

```
$ fglmkrun
```

This command creates the default P-code runner, called **fglrun**, in the **\$FGLDIR/bin** directory.

If you need your own, statically-linked runner, use the syntax in the following example (assume your runner is named **myrun**, you are using Informix Client SDK version 2.10, and using a C function file named **file.c**):

```
$ fglmkrun -d ix914 -add -static $FGLDIR/lib/fglExt.c
                                     file.c -o myrun
```

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

```
$ rtsinstall
```

For details about **fglmkrun**, see [“Details About fglmkrun” on page 4-11](#).

SCO Systems

With SCO systems, the use of **fglmkrun** during a manual installation might 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** that contains the following code:

```
#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 **fglrun** in the current directory.

Creating the C-Code Libraries

If you have Version 6.x or Version 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 UNIX.

Interruption Signal

When you press the interrupt key or the **Interrupt** button, your client computer intercepts this and sends 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.

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 computer 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 the 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 message 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 on UNIX.

Finding Informix 7.x Libraries on UNIX

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 ESQ/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:

```
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**:

```
main(argc,argv)  
int argc; char *argv[];  
{  
}
```

Compile it using your copy of ESQL:

```
$ ./esql t.ec
```

This gives you the compile statement with all the libraries used on the standard output. For example:

```
cc -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 /usr1/informix7.11/lib/esql/libosshr.a  
-L /usr1/informix7.11/lib /usr1/informix7.11/lib/libasfshr.a -L  
/usr1/informix7.11/lib/esql  
/usr1/informix7.11/lib/esql/libgenshr.a -L  
/usr1/informix7.11/lib/esql /usr1/informix7.11/lib/esql/libosshr.a  
-L /usr1/informix7.11/lib/esql  
/usr1/informix7.11/lib/esql/libgenshr.a -ltli -L  
/usr1/informix7.11/lib /usr1/informix7.11/lib/libnetstubshr.a -lc  
-lmsaa -lbsd
```

In this case, you would have to set your environment variable **FGLLIBSQL** like this:

```
$ 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 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 UNIX

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:

```
$ 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
```

and the first number is greater than zero (here it is 1640), the library **libname.a** must be added to the list of the system libraries needed to create a P-code runner or a C-code application. This list is specified by the **FGLLIBSYS** environment variable and is built in the same way as **FGLLIBSQL**.

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 database 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 U.S. English locale (**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 see the supported versions.

Workarounds for the Windows Client

This section describes workarounds for Windows Client problems.

Slow rlogin and Some Characters Appear Twice

With WTK, you can have a 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 Client, 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 Client in a directory that does not conform to the 8.3 naming convention. For example, you cannot install the package in a directory named **directoy~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 computer. You can check if the file **winsock.dll** is in the **Windows** directory. If not, install the TCP/IP socket support on your client computer.

4GL Program Errors

This section describes workarounds to use if you experience errors with your 4GL program.

Internal Data File Corrupted

On some UNIX systems (for example, SCO), you might receive the following error message:

```
Internal data file corrupted. Cannot continue this program.
```

After that, your program fails.

This failure occurs 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 superuser.

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:

```
FGLPS="ps -aefx"
export FGLPS
```

If there is no command that allows a non-superuser to view the whole process list on the operating system, you can use the following procedure (you need a C compiler installed on your computer):

- **Log in as root:**

```
$ cd $FGLDIR/src
```

- **Edit the file `psall.c` and change the «`ps -ae`» command if needed:**

```
$ 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:

```
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 Computers

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                CONFIGALLOC 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  2115  40  0  
data block size128  2048  54  1994  242  57  0  
data block size256  1024  15  1009  55  17  0  
data block size512  256   0   256   28   1  0  
data block size1024  452   0   52    9   1  0  
data block size2048  850   0   50   14   1  0  
data block size4096  641   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**:


```
int __write(int fd, char *c, int l) {
    return(write(fd,c,l));
}
```
- 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.

Key Buttons Missing

If you are using an SCO system and no key buttons appear, change the dot (.) to a comma (,) in the In **/usr/lib/lang/\$LANG/*/numeric** file.

To test, call **wish** or **tcsh**, 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 a text 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
```

Workarounds for X11

This section describes various other workarounds that you will find helpful.

No Program Display at Startup

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, the problem is usually 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 displays lines similar to the following:

```
shift Shift_L(0x32), Shift_R(0x3e)
lock Caps_Lock(0x42)
controlControl_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.

To enable the numeric keypad with the **Num_Lock** key disabled, specify the following lines in the file `$HOME/.Xmodmap`:

```
keycode 63 = KP_Multiply
keycode 79 = KP_7
keycode 80 = KP_8
keycode 81 = KP_9
keycode 82 = KP_Subtract
keycode 83 = KP_4
keycode 84 = KP_5
keycode 85 = KP_6
keycode 86 = KP_Add
keycode 87 = KP_1
keycode 88 = KP_2
keycode 89 = KP_3
keycode 90 = KP_0
keycode 91 = KP_Decimal
keycode 108 = KP_Enter
keycode 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 on your keyboard layout.

CapsLock and Scrollbar

If **CapsLock** is on, the scrollbar does not work in GUI mode with the X11 front end.

Workarounds for Windows

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 Windows NT

Never change or edit `fgl2c.tcl` on Windows NT. If you do that, **^M** characters are 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 computer having Ataman remote login services installed, perform the following steps:

- 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 occurs 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, enter the following command:

```
$ rsh PC_name "winexec progname.exe"
```

From your UNIX system, enter the following command:

```
$ 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 computers, you should not use the **emm386** memory manager.

Error Messages

This appendix lists error messages and suggested solutions for the following kinds of errors:

- Form compilation errors
- 4GL compilation errors
- Runtime errors
- UNIX X11 client errors
- License errors
- **fglmkrun** errors

Form Compilation Errors

- 1312 FORMS statement error number <number>.
- Description.* An error occurred in the form at runtime.
- Solution.* Edit your source file, go to the specified line, correct the error, and recompile the file.
- 1314 Program stopped at <filename>, line number <number>.
- Description.* At runtime an error occurred in the specified file at the specified line. No **.err** file is generated.
- Solution.* Edit your source file, go to the specified line, correct the error, and recompile the file.
- 1320 A function has not returned the correct number of values expected by the calling function.

- 2975 **Description.** A function that returns several variables has not returned the correct number of parameters.
Solution. Check your source code and recompile.
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.
- 6601 fglform: Cannot open database dictionary <filename>. 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 environment variable **FGLDBPATH**.
- 6802 fglform: Cannot open Database dictionary '<name>'. Run **fglschema** database.
Description. The form compiler cannot find the schema of the specified database.
Solution. Check if the schema of the database exists and check if the **FGLDBPATH** environment variable is well set to the path to the schema.
- 6805 Open Form <name>', Bad Version:<number>, expecting: <number>.
Description. The form that you are trying to open has been compiled with an old version of the compiler.
Solution. Recompile your form with the new form compiler.

4GL Compilation Errors

- 4900 fglcomp: This syntax is not supported here. Use [screen record name.] screen field name.
- 4901 fglcomp: Fatal INTERNAL error: <fieldname>.
- 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.
- 6011 Demonstration version.
- Description.* This message is displayed only by the demonstration version.
- Solution.* This message is for informational purposes only.
- 6020 Installation: Cannot open <filename>.
- 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.
- 6023 C-code generation is not allowed with the demonstration program.
- Description.* Dynamic 4GL can compile in P code and in C Code (only for the UNIX version). But with the demonstration version, C-code compilation is not available.
- Solution.* Compile your program in P code.
- 6601 Cannot open database dictionary <filename>. 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.

- 6602 Cannot open globals file <filename>.
- 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.
- 6603 The file <filename> 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.
- 6605 The module <name> does not contain function <name>.
- 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.
- 6606 No member function <name> for class <name> 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.
- 6607 Wrong number of dimensions for <array>.
- 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.
- 6608 Resource error: <number>: parameter expected.
- Description.** An unexpected error occurred.
- Solution.** Contact Informix Technical Support.

Runtime Errors

- 1310 Program error at <filename>, line number <number>.
- 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: <date> Time: <time>
- Description.* This is internal information.
- Solution.* No solution is required.
- 6300 Cannot connect to a 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 computer. This is the computer that executes the fglX11d daemon for a UNIX system or the 4GL server for Windows systems. **DISPLAY** must be set on the client computer. For Windows, this variable cannot be set. Also check if the graphical daemon is running.
- 6301 Cannot write to the 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 computer. This is the computer that executes the fglX11d daemon for a UNIX system or the 4GL server for Windows systems. **DISPLAY** must be set on the client computer. For Windows, this variable cannot be set. Also check if the graphical daemon is running.
- 6302 Cannot read from the 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 computer. This computer executes the fglX11d daemon for a UNIX system or the 4GL server for Windows systems. **DISPLAY** must be set on the client computer. For Windows, this environment variable cannot be set. Also check if the graphical daemon is running.
- 6303 Wrong script (fgl2c.tcl) version. Check installation.
- Description.** The graphical daemon has loaded a version of the client different from the one defined in the resource files of the current version as defined by the **\$FGLDIR** environment variable.
- Solution.** Stop and restart the graphical daemon each time you change the graphical client.
- 6304 Wrong server (wtkclt) version. Check installation.
- Description.** The graphical daemon has loaded a version of the client different from the one defined in the resource files of the current version as defined by the **\$FGLDIR** environment variable.
- Solution.** Stop and restart the graphical daemon each time you change the graphical client.
- 6306 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.
- 6307 Server autostart: cannot identify workstation.
- Description.** You must set the **FGLSERVER** environment variable, as well as the entry of the autostart feature in the **\$FGLDIR/etc/fglprofile** file.
- Solution.** Set the needed environment variables or add values in the **\$FGLDIR/etc/fglprofile** file to enable the graphical daemon autostart feature.
- 6308 Server autostart: unknown workstation: set fglrun.server.<number> = <aliaslist>.
- Description.** The computer described by the entry fglrun.server.## in the **fglprofile** file is not accessible on the network.

- Solution.** Check if the computer name is correctly set in the **DISPLAY** or **FGLSERVER** environment variable.
- 6309 Not connected. Cannot write to the GUI.
- Description.** The communication between the 4GL application and the graphical client is broken.
- Solution.** Check if the **\$FGLSERVER** and the **\$DISPLAY** variables are correctly set. Also check if the daemon of the graphical front end is running.
- 6310 Not connected. Cannot read from the GUI.
- Description.** The communication between the 4GL application and the graphical client is broken.
- Solution.** Check if the **\$FGLSERVER** and the **\$DISPLAY** environment variables are correctly set. Also check if the daemon of the graphical client is running.
- 6320 Cannot open file <filename>.
- Description.** The compiler cannot access the resource file **\$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.
- 6321 No such interface capability: <filename>.
- Description.** The resource files from the graphical client are from different versions. This is often caused by installing an update over an old version of the compiler. Because of permission problems, some files have been overwritten while others have not.
- Solution.** Check the permissions of the files located in the **\$FGLDIR** directory and re-install the update.
- 6322 <version number> wrong version. Expecting <version number>.
- 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.

Runtime Errors

- 6323 Cannot open factory profile <filename>.
- Description.* The `$FGLDIR/etc/fglprofile` file is missing or is unreadable.
- Solution.* Check the permission of the file. If the file is missing, reinstall the compiler.
- 6324 Cannot load customer profile <filename>.
- Description.* The configuration file defined by the `FGLPROFILE` environment variable is missing or unreadable.
- Solution.* Check if the `FGLPROFILE` environment variable is correctly set and if the file is readable by the current user.
- 6325 Cannot load application resources <name>.
- Description.* The directory specified by the `fglrun.default` entry in `$FGLDIR/etc/fglprofile` is missing or not readable for the current user.
- Solution.* Check if the entry `fglrun.default` is correctly set in `$FGLDIR/etc/fglprofile` and if the directory specified is readable by the current user.
- 6305 Cannot open char table file. Check your **fglprofile**.
- Description.* This error occurs if the conversion file defined by the `gui.chartable` entry, in the `$FGLDIR/etc/fglprofilefile`, 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.
- 6327 Internal error in the runtime library file <name>.
- Description.* Something unpredictable occurs, generating an error.
- Solution.* Contact Informix Technical Support.
- 6340 Cannot open file.
- 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.

- 6341 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.
- 6342 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.
- 6343 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.
- 6344 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.
- 6345 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.
- 6346 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.

- 6200 fglrun: Module <name>: The function <name> will be called as <name>.
Description. An incorrect number of parameters are used to call a 4GL function.
Solution. Check your source code and recompile your application.
- 6201 fglrun: Module <name>: 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.
- 6202 fglrun: File <name>: Bad magic number: Code cannot run with this P-code computer.
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.
- 6203 fglrun: Module <name>: The function <name> has already been defined in module <name>.
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.
- 6204 fglrun: Module <name>: Unknown P code.
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.
- 6205 fglrun: INTERNAL ERROR: Alignment.
Description. This error is internal, which should not normally occur.

- Solution.** Contact Informix Technical Support.
- 6206 fglrun: The dynamic loader cannot open module '<name>'.
Description. The module is not in the current directory or in one of the directories specified that the environment variable **FGLLDPATH** specifies.
Solution. Set the environment variable **FGLLDPATH**.
- 6207 fglrun: The dynamic loaded module '<name>' does not contain the function '<name>'.
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 you execute your application.
- 6208 fglrun: Module '<name>': 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.
- 6209 fglrun: 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. Check your environment.
Description. When a client computer starts an application on the server, the application stores data in the **\$FGLDIR/lock** directory. 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 This demonstration version allows one user only.
Description. The demonstration 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 <name>.
- 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.
- 6022 Demonstration time has expired. Run this program again.
- Description.** The runtime demonstration version is valid only for a few minutes after you have started a program.
- Solution.** Restart the program.
- 6023 C-code generation is not allowed with the demonstration 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 demonstration version.
- Solution.** Compile your program in P code.
- 6025 Demonstration time has expired. Contact your vendor.
- Description.** The demonstration version of Dynamic 4GL has a time limit of 30 days.
- Solution.** Either reinstall a new demonstration version or call your Dynamic 4GL distributor.
- 6026 Bad link for runner demonstration. Retry or rebuild your runner.
- Description.** The runner is corrupted.
- Solution.** Relink your runner with the `fglmkrun` tool.
- 6362 Unknown user name. Set the 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 environment 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.
- 6328 Bad format of resource %s value %s : you must use next syntax
%s='VARNAME=value'.
- Description.** In the `$FGLDIR/etc/fglprofile` file, a `fglrun.setenv.x` or a `fglrun.defaultenv.x` entry is incorrectly set.
- Solution.** Check your configuration file and correct the error.
- 6329 Cannot put in process environment the next variable: '%s'
- Description.** A variable defined in the `$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.
- 6363 The `INFORMIXDIR` environment variable is not set. Check your environment.
- Description.** The `INFORMIXDIR` environment variable is not set. This value is required by the Dynamic 4GL compiler.
- Solution.** Set the environment variable to the name of the directory where the Informix products are installed.
- 6211 Link has failed.
- Description.** A problem occurred while linking the runner.
- Solution.** Check if all the variables have been set and retry.
- 6326 Cannot open char map file '<name>'. Check your **fglprofile**.
- Description.** The specified char map file cannot be found or read.
- Solution.** Verify that the char map file is located in `$FGLDIR/etc`, and that the right value is set in `fglprofile` (`GUI.CHARTABLE` entry).
- 6360 This runner cannot execute any SQL.
- Description.** You are trying to run a program that contains some SQL statements with the `fglnodb` runner (unable to access any database)
- Solution.** Use the full-featured runner.

- 6361 Dynamic SQL: type unknown: <type>.
Description. The specified type is unknown for the database.
Solution. The known types can be CHAR, VARCHAR, INTEGER, SMALLINT, FLOAT, SMALLFLOAT, DECIMAL, MONEY, DATE, DATETIME, INTERVAL.
- 6604 (obs) The function 'fgl_dialog_<name>' can only be used within an INPUT [ARRAY], DISPLAY ARRAY, or CONSTRUCT statement.
Description. You can only use this function on an "on key" statement (with an INPUT [ARRAY], DISPLAY ARRAY or CONSTRUCT statement.)
Solution. In the rest of the program, use the other functions without the fgl_dialog_ prefix.
- 6610 The function '<name>' has already been called with a different number of returned values.
Description. A function, not defined in the same module, has been called two times with two different numbers of parameters.
Solution. This is a mistake in the source code. Check the function definition and correct the wrong call.
- 6611 Function '<name>': unexpected number of returned values.
Description. You are calling a function that returns a different number of values than you expected at call.
Solution. Check your source.
- 6612 Redclaration of function '<name>'.
Description. A function has been defined twice in your program/module.
Solution. Check if you do not have two different functions with the same name.
- 6613 The library function '<name>' is not declared.
Description. You are calling a function that is not defined.
Solution. Check your sources.
- 6614 The function '<name>' might return a different number of values.

Description. When you are using the `-W` return flag in compilation command, this message warns you.

Solution. Only a warning.

-6615

The symbol '<name>' is unused.

Description. When you are using the `-W unused` flag in compilation command, this message warns you that you have unused variables.

Solution. Only a warning.

UNIX X11 Client Errors

-6502

`fglX11d (%d)` could not be stopped.

Description. Problem of rights (process owner, ...).

Solution. Check if you are trying to shut down an application that does not belong to you.

-6505

Communication between daemon and interface manager has broken down.

Description. The dedicated `wish` program is not available as it should be.

Solution. Check if the location of `wish` can be found in the `PATH` environment, if it can be launched by the current user, and if it is the correct 4js version. (You should check by running `wish -v`.)

License Errors

-6012

Cannot get license information. Check your environment and the license (run "`fglWrt -a see`").

Description. You might have a different value between the `FGLDIR` environment variable and the path to the Dynamic 4GL binary files defined in the `PATH` environment variable.

- Solution.** Set the **FGLDIR** environment variable and 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%`
- 6013 Time-limited version: time has expired.
- Description.** You have installed a demonstration 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 demonstration 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 Cannot get license information.
- 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 Cannot get information for license (Error %s). Check your environment and the license (run "fglWrt -a see").
- Description.** The application is unable to check the license validity.
- Solution.** You must have read permissions on all the files and write permissions 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 license manager.
- Check the following:
- **fgllic.server** entry in **fglprofile**
 - the license manager host
 - the license manager program
- Description.** You have not specified a value for the environment variable **fgllic.server** in the **\$FGLDIR/etc/fglprofile** file.
- Solution.** Check the **fglprofile** file for the entry point **fgllic.server** and specify the name (in uppercase letters) of the computer that runs the Dynamic 4GL License Server.
- 6029 Unknown parameter '<name>' for checking.
- Description.** There is a wrong parameter on the command line of the **fglWrt** tool.
- Solution.** Check your command-line parameters and retry the command.
- 6030 The -J option requires the license number before the license key.
- Description.** You used the **fglWrt** program with the -J flag but you entered an incorrect serial number or the license key before the serial number.
- Solution.** Specify a valid serial number.
- 6031 Temporary license has expired.
- Description.** Your temporary runtime license has expired.
- Solution.** Call your Dynamic 4GL distributor to get a new license.
- 6032 <name>: illegal option: <name>
- Description.** The specified program (<name>) has been called with a specified parameter (<name>) that is not recognized by the program.
- Solution.** Run the program using the flag -h or -h to get help information about it.

- 6033 <name>: '<parameter>' option requires an argument.
Description. You cannot use this option of the fglWrt tool without a parameter.
Solution. Check your command line and try the command again.
- 6034 Warning! This is a temporary license, installation number is '%s'.
Description. You have installed a temporary license of 30 days. You will have to enter an installation key before the end of this period if you want to keep on running the program.
Solution. This is only a warning message.
- 6035 Cannot read in 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.
- 6041 Problem while searching license information.
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 Incorrect license information. Verify if a license is installed.
Description. You have attempted to run Dynamic 4GL 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 The testing period is finished. You must install a new license.
Description. The test time license of Dynamic 4GL has expired.
Solution. Call your Dynamic 4GL distributor to purchase a new license.

- 6044 Incorrect information in license program. Verify if a license is installed or check if your are on the right computer when a license manager is used.
- 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.
- (First verify that you have installed theDynamic 4GL license. For the Dynamic 4GL License Server, check that you are on the right computer.)
- 6045 Incorrect information in license program. Verify if a license is installed or check if your are on the right computer when a license manager is used.
- 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.
- (First verify that you have installed theDynamic 4GL license. For the Dynamic 4GL License Server, check that you are on the right computer.)
- 6046 Cannot read license information. Check **FGLDIR** and your environment.
- Description.** Several environment variables must be set correctly.
- Solution.** Check your environment variables. Check your license by running the program `fglWrt -a see`.
- 6047 Incorrect information in license program. Verify if a license is installed.
- 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.
- (First verify that you have installed theDynamic 4GL license. For the Dynamic 4GL License Server, check that you are on the right computer.)
- 6048 Incorrect information in license program. Verify if a license is installed.

- 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.
- (First verify that you have installed the Dynamic 4GL license. For the Dynamic 4GL License Server, check that you are on the right computer.)
- 6049 This product is licensed for runtime only. No compilation is allowed.
- 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 license expired. Contact your vendor.
- 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.
- 6051 Temporary license expired. Contact your vendor.
- 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 license expired. Contact your vendor.
- 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 The **FGLDIR** environment variable has changed. **FGLDIR** must hold the original installation path.
- Description.** The value of **FGLDIR** or the location of **FGLDIR** has been changed.

- 6054 **Solution.** Ask the person who installed the product for the location of the original installation directory and then set the **FGLDIR** environment variable.
- Cannot read a license file. Check **FGLDIR** and your environment. Verify if a license is installed.
- Description.** The file that contains 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 update a license file. Check **FGLDIR** and your environment. Verify if a license is installed.
- Description.** The file that contains 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**.
- 6056 Cannot write into a license file. Please check your rights.
- Description.** The file that contains 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. Verify if a license is installed.
- Description.** The file that contains 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 Incorrect license file format. Verify if a license is installed.
- Description.** The file that contains 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 Incorrect license file format. Verify if a license is installed.
- Description.** The file that contains the license has been corrupted.

- 6068 **Solution.** Reinstall the license. If you have a backup of the current installation of Dynamic 4GL, restore the files located in the `$FGLDIR/etc` directory.
- No license installed.
- Description.** There is no license installed for Dynamic 4GL.
- Solution.** Install a license. If a license is already installed, check that the `$FGLDIR` environment variable is set correctly.
- 6069 Cannot uninstall the license.
- Description.** There was a problem during the uninstall of the Dynamic 4GL license.
- Solution.** Check if the `FGLDIR` environment variable is correctly set in your environment and if the current user has permission to delete files in the `$FGLDIR/etc` directory.
- 6070 The `fgllic.server` entry must be set in `fglprofile` in order to reach the license manager.
- Description.** You are using the remote license process and you have set the value of `fgllic.server`, in `$FGLDIR/etc/fglprofile`, to localhost or to the 127.0.0.1 address.
- Solution.** You must use the real IP address of the computer even if it is the local computer.
- 6071 Cannot use directory '<name>'. Check `FGLDIR` and verify if access rights are 'drwxrwxrwx'.
- 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 `FGLDIR` and verify if access rights are 'drwxrwxrwx'.
- 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 '%s'. Verify if access rights are 'drwxrwxrwx'.
- Description.* The compiler needs to make an operation in the specified directory.
- Solution.* Change the permission of this directory to 777 mode.
- 6074 '<name>' does not have 'rwxrwxrwx' rights or is not a directory. Check access rights with "ls -ld \$FGLDIR/lock" or execute "rm -r \$FGLDIR/lock" if no users are connected.
- 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 '<name>'. Check **FGLDIR** and verify if access rights are 'drwxrwxrwx'.
- 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. 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 SYSERROR. Cannot set socket to non-blocking mode. Check the system error message and retry.
- Description.* When starting an application, a problem occurs with the initialization of the socket of the Windows computer.
- Solution.* Restart the program. If the problem still exists, check that the TCP/IP stack is correctly installed and configured on your computer.
- 6079 Cannot get computer name or network IP address. Each network client must have an IP address when using a license manager. **FGLSERVER** must hold the IP address or the host name of the client (localhost is not allowed).
- 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 computer even if it is the local computer. This is also true for the value used with the **FGLSERVER** environment variable.
- 6080 Cannot get information from host %s. Check '**fgllic.server**' entry in **fglprofile**.
- 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 reach host %s with ping: Check '**fgllic.server**' entry in **fglprofile**. Check your network configuration or increase '**fgllic.ping**' value.
- Description.* The license server cannot ping the client computer, 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 computer. 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. Check the system error message and retry.

- Description.** There is a problem with the socket of the Windows computer.
- Solution.** Check that the system is correctly configured and retry the program.
- 6083 SYSEERROR(%d)%s: Cannot set option DONTLINGER on socket. Check the system error message and retry.
- Description.** There is a problem with the socket of the Windows computer.
- Solution.** Check that the system is correctly configured and retry the program.
- 6084 SYSEERROR(%d)%s: Cannot set option LINGER on socket. Check the system error message and retry.
- Description.** There is a problem with the socket of the Windows computer.
- Solution.** Check that the system is correctly configured and retry the program.
- 6085 SYSEERROR(%d)%s: Cannot connect to the license manager on host '%s'. Check the following places:
- fglic.server entry in **fglprofile**
 - the license manager computer
 - the license manager service
- 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 computer where the product is installed.
- Solution.** Check that the Dynamic 4GL License Server is running on the computer where the product is installed.
- 6086 SYSEERROR(%d)%s: Cannot send data to the license manager. Check the system error message and retry.
- Description.** There is a problem with the socket of the Windows computer.
- Solution.** Check that the system is correctly configured and rerun the program.
- 6087 SYSEERROR(%d)%s: Cannot receive data from license manager.
- Check the system error message and retry.
- Description.** There is a problem with the socket of the Windows computer.

-6088	<p>Solution. Check that the system is correctly configured and rerun the program.</p> <p>You are not allowed to connect for the following reason: %s</p> <p>Description. The program cannot connect to the license server because of the specified reason.</p> <p>Solution. Try to fix the problem described and rerun your application.</p>
-6090	<p>SYSEERROR(%d)%s: Cannot create a socket to start the license manager. Check the system error message and retry.</p> <p>Description. There is a problem with the socket of the Windows computer.</p> <p>Solution. Check that the system is correctly configured and rerun the program.</p>
-6091	<p>SYSEERROR(%d)%s: Cannot bind socket for the license manager. Check the system error message and retry.</p> <p>Description. There is a problem with the socket of the Windows computer.</p> <p>Solution. Check that the system is correctly configured and rerun the program.</p>
-6092	<p>SYSEERROR(%d)%s: Cannot listen socket for the license manager.</p> <p>Description. There is a problem with the socket of the Windows computer.</p> <p>Solution. Check that the system is correctly configured and rerun the program.</p>
-6093	<p>SYSEERROR(%d)%s: Cannot create a socket to search an active client.</p> <p>Description. There is a problem with the socket of the Windows computer.</p> <p>Solution. Check that the system is correctly configured and rerun the program.</p>
-6094	<p>SYSEERROR(%d)%s: This is a WSASStartup error. Check the system error message and retry.</p> <p>Description. There is a problem with the socket of the Windows computer.</p> <p>Solution. Check that the system is correctly configured and rerun the program.</p>

- 6095 Cannot start the license manager: %s
Description. License type incompatible. You are installing a version with an unappropriated license (There is a license server and you are installing a version of a classical one)
Solution. Reinstall and then contact your vendor.
- 6096 Connection refused by the license server.
Description. There is problem connecting the client computer to the Windows license server.
Solution. The problem is due to a configuration problem of the license server computer. Check the configuration of the computers and of the products.
- 6098 Stopping the license manager.
Description. The license server service is stopping.
Solution. This is an informational message.
- 6099 SIGTERM received. Stopping the license Manager.
Description. The license server service is stopping.
Solution. This is an informational message.
- 6107 User limit exceeded. Retry later.
Description. The maximum number of clients that can be run has been reached (due to the license installed).
Solution. Retry later (when the number of current users has decreased) or install a new license that allows more users.
- 6108 Environment is incorrect.
Description. There is no local license or the environment is not set right.
Solution. Check your environment and your **FGLDIR** environment variable.
- 6109 Cannot add session #%s.
Description. You do not have the rights to create the new session (in fact the directory representing the new client).
Solution. Check the rights of the dedicated directories.

License Errors

- 6110 Cannot add program '%s' (pid=%d).
Description. You do not have the rights to create the new application for the current user (in fact the file representing the new application).
Solution. Check the rights of the dedicated directories.
- 6114 Cannot start program '%s'.
Description. When using fglWrt -u to find the number of users allowed on this installation, the command "ps" used can be launched (only for UNIX).
Solution. Check the rights for ps.
- 6148 The FGLDIR environment variable is not set.
Description. You are handling licenses but the FGLDIR environment variable is not set.
Solution. Set the FGLDIR environment variable and retry.
- 6149 Problem while installing license '%s'.
Description. A problem occurred while licensing.
Solution. Note the system-specific error number and contact Informix Technical Support.
- 6150 Temporary license not found for this version.
Description. While adding a definitive license key, the assumed temporary license has not been found.
Solution. Re-install the license.
- 6151 Wrong installation key.
Description. While adding a definitive license key, the found installation key was not valid.
Solution. Re-install the license.
- 6152 Problem during license installation.
Description. A problem occurred while installing the license. Could not write information to the disk (either own files or system files).

- Solution.** Check the **FGLDIR** environment variable and the rights of the license files (must be able to change them).
- 6155 This license is too old to be valid.
- Description.** The temporary license time has expired.
- Solution.** You have to install a new license.
- 6156 Too many temporary licenses.
- Description.** You installed a temporary license too many times.
- Solution.** Contact technical support to get a valid license.
- 6158 Cannot store temporary information.
- Description.** A problem occurred while installing the license. Could not write information to the disk (either own files or system files).
- Solution.** Check the **FGLDIR** environment variable and the rights of the license files (you must be able to change them).
- 6162 A valid license is already installed.
- Description.** You are trying to install a new license on an existing one.
- Solution.** You should not overwrite an existing license (or delete the existing first).
- 6168 Problem (b1) during license installation.
- Description.** The program cannot find out some system information (about the program's owner).
- Solution.** Check your installation.
- 6190 %s is already installed.
- Description.** A license server is already installed.
- Solution.** You should not try to overwrite an existing license server.
- 6192 %s installation failed. Error %d.
- Description.** A problem occurred while installing the license server.
- Solution.** Check your environment.

-6193 %s is not installed.
Description. You are trying to uninstall the license manager which is not installed.

-6195 Could not remove %s. Error %d.
Description. An error occurred while uninstalling the license manager.
Solution. Note the system error and contact your support center.

Licensing Problems (Windows NT)

-6701 Cannot access the service.
Description. The relevant rights to launch the license service are not allowing it to be started.

Solution. Check your system configuration.

-6702 Internal error.
Description. Internal error of memory allocation.

Solution. None: Retry.

-6703 The service binary file could not be found.
Description. The path to the service is wrong.
Solution. Verify if it has been installed properly or is present in the <FGLDIR>\bin.

-6704 The registry database is locked.
Description. You cannot access the registry database's information because it is locked.

Solution. Check your system configuration.

-6705 The service depends on a service that does not exist or has been marked for deletion.
Description. The license service cannot be launched because it depends on an unexisting service (which has probably not been installed).

- 6706 **Solution.** Look for the missing service.
The service depends on another service that has failed to start.
Description. The license service cannot be launched because it depends on another service that cannot be run.
Solution. Retry.
- 6707 A thread could not be created for the Win32 service.
Description. A thread could not be created while trying to run the license server.
Solution. Retry.
- 6708 The requested control code is not valid, or it is unacceptable to the service.
Description. While trying to shut down the license server, the service got a bad instruction code.
Solution. Call technical support.
- 6709 %s is not installed.
Description. You are trying to install the server and it failed, or you were trying to uninstall it, but it could not be found.
Solution. Re-install the license server using the command 'fglserv -i'.
- 6710 %s does not exist.
Description. The host you are using as a license server has no specific service running to afford this.
Solution. Check if your fgllc.server entry in your **fglprofile** is right set.
- 6712 %s is already running.
Description. You are trying to run a service that is already running.
Solution. Do not try to start it.
- 6714 %s could not be logged on.
Description. The service cannot be started logging on as the specified user.

- 6715 **Solution.** Check the service's properties (if the password has changed and so on).
%s has been marked for deletion.
Description. The service has been marked for deletion so it cannot be used anymore (will be deleted at next shutdown of the computer).
Solution. Re-install the license server.
- 6716 %s did not respond to the start request in a timely fashion.
Description. One of the service's component could not be started and timed out.
Solution. Call your technical support.
- 6717 An error occurred while starting %s.
Description. An error occurred while starting the service.
Solution. Retry.
- 6719 %s is not correctly installed.
Description. The license service is not correctly installed.
Solution. Re-install the license server.
- 6720 %s was not opened with the necessary access.
Description. The sufficient rights to stop the server are not completed.
Solution. Check your environment and your right access.
- 6721 %s cannot be stopped because other running services are dependent on it.
Description. The service cannot be stopped because another one needs it.
Solution. This error should not occur in any case for the moment: if it occurs, there is a problem with your system.
- 6722 The requested control code cannot be sent to %s because the state of %s is not correct.
Description. The service is in a mode where it cannot be requested anymore (for the moment).

- 6723 **Solution.** Retry.
%s has not been started.
Description. You are trying to shut down a server that is not running.
Solution. Do not try to stop this service.
- 6724 %s did not respond to the start request in a timely fashion.
Description. A service does not respond anymore.
Solution. Call the support team.
- 6725 %s generated an error at stopped time.
Description. An error occurred during the shutdown time.
Solution. None: retry.
- 6727 %s is not correctly installed.
Description.**Description.** The service does not exist or has not been installed properly.
Solution. Install or re-install it.
- 6729 %s not correctly started.
Description.**Description.** Installation process creation has failed.
Solution. Retry.
- 6731 %s is not correctly uninstalled.
Description. Uninstallation process creation has failed.
Solution. Retry.
- 6733 Unable to uninstall the service.
Description. Cannot uninstall the service.
Solution. Check if it is running if you have the rights.
- 6750 Cannot open **FGLDIR** directory tree.
Description. The **FGLDIR** tree is missing or the **FGLDIR** variable is not set.

-6751

Solution. Check your environment.

This license is available only on a Windows NT computer.

Description. You are trying to license with a WLS license type on a Win95 or a Win98 station.

Solution. You have to use Windows NT to do that or to use a local license.

fglmkrun Errors

The following list shows **fglmkrun** error messages and solutions. These error messages have no error message number.

- -o flag must be followed by a name.

Description. You added the -o flag but did not include the name of the runner to be created.

Solution. Add the name of the runner after the -o flag. For example, \$ fglmkrun -o myrun.

- -d flag must be followed by a database interface type.

Description. You added the -d flag but did not include the database interface type.

Solution. Add the database interface type after the -d flag. For example \$ fglmkrun -d ix914.

- -sh flag must be followed by a program name.

Description. You added the -sh flag but did not include the shell name to compile the P-code runner.

Solution. Add the shell name after the -sh flag. For example, \$ fglmkrun -sh esql.

- -add flag must be followed by one argument. For example, \$fglmkrun-add-static.

Description. You added the -add flag but did not specify any flag to send to the tool compiling the P-code runner.

Solution. Add the flag to send the tools compiling the P-code runner after the -add flag.

- You should use "esql" or "c4gl" to create a runner when using -d ixgen.

Description. When using the d-ixigen flag (i.e.: using the generic database interface), you must use the "esql" or the "c4gl" script to create the runner. You cannot use a standard C compiler to compile an ESQL/C file.

Solution. Use the "esql" or the "c4gl" script to create the runner (\$FGLDIR/src/esql_gen.ec).

- Could not find database interface library for ixNNN.

Description. In this message, ixNNN is the parameter added after the -d flag. This string is used to build the name of the libraries used to build the runner. For example, when you use -d ix914, you use the library named \$FGLDIR/lib/libix914.a.

Solution. Enter a valid parameter name to access the appropriate library. The only allowed parameters are ix410, ix501, ix711, ix914 or ixgen.

- XXX was not found or cannot be executed. Please check your environment.

Description. The shell specified by the -sh flag (in this case XXX) cannot be found in any of the directories specified by the PATH environment variable, or it does not have execution permission.

Solution. Ensure that the PATH variable specifies the directory where the shell is located, and that execution permissions are properly set.

- Could not create runner XXX. Please check the following error messages: <followed by other error messages>.

Description. The runner named XXX cannot be created. This could be due to errors located in the environment, or it could be due to the manner in which various products (ESQL, Dynamic 4GL, database libraries, C compiler, and so forth) are installed.

Solution. Check the environment for possible errors. Then run the fglmkrun command again using the -vb verbose flag to help identify the problem.

Global Language Support

This appendix describes the Global Language Support (GLS) feature available in Dynamic 4GL. The GLS feature allows Informix database servers to handle different languages, cultural conventions, and code sets.

This appendix describes the GLS features unique to Dynamic 4GL. You should be familiar with using GLS features and GLS behaviors before using this appendix.

For Additional GLS Information

For additional information on using GLS, refer to the following Informix guides:

- *GLS Programming Guide* included with the Informix Dynamic Server 7.2x (or greater) documentation.
- *Informix Guide to GLS Functionality* included with the Informix Dynamic Server 7.2x (or greater) documentation.

Informix guides are available from the Informix Online Documentation web site. To access this web site, use the following URL:

<http://www.informix.com/answers>

All manuals listed on the Web site are stored in Adobe Acrobat (.pdf) format.

Software Requirements

You do not need 4GL installed to install Dynamic 4GL. However, the Client SDK, Version 2.x or later, must be installed.

The Client SDK installs the latest version of GLS. To use the GLS feature in Dynamic 4GL, you must be using Informix GLS 3.07 (or later).

Displaying the GLS Version

You can display the version of GLS you currently have installed. To display the GLS copyright message, type the following:

```
cat $INFORMIXDIR/etc/GLS-cr
```

The following text appears:

```
INFORMIX LIBGLS LIBRARY Version 3.08.UC1  
Copyright (C) 1991-1998 Informix Software, Inc.
```

Downloading the Client SDK

The latest version of the Client SDK is available to download from the Informix Web site at the following URL:

```
http://www.intraware.com/informix/
```

Supported Dynamic 4GL Clients

The only clients able to run Dynamic 4GL applications with the GLS feature enabled are:

- Windows Client
- Text (ASCII) Client
- Java Client

The following Dynamic 4GL clients do not support all languages:

- X11 Client
- HTML client

Database Server Compatibility

This section describes the compatibility of Dynamic 4GL with Informix database servers and other Informix products.

To use the GLS features of Dynamic 4GL, any database or connectivity products must support the Informix GLS library, Version 3.07 or higher. [Figure D-1](#) summarizes the relationships of Informix Dynamic 4GL to the UNIX-based Informix database servers that it supports.

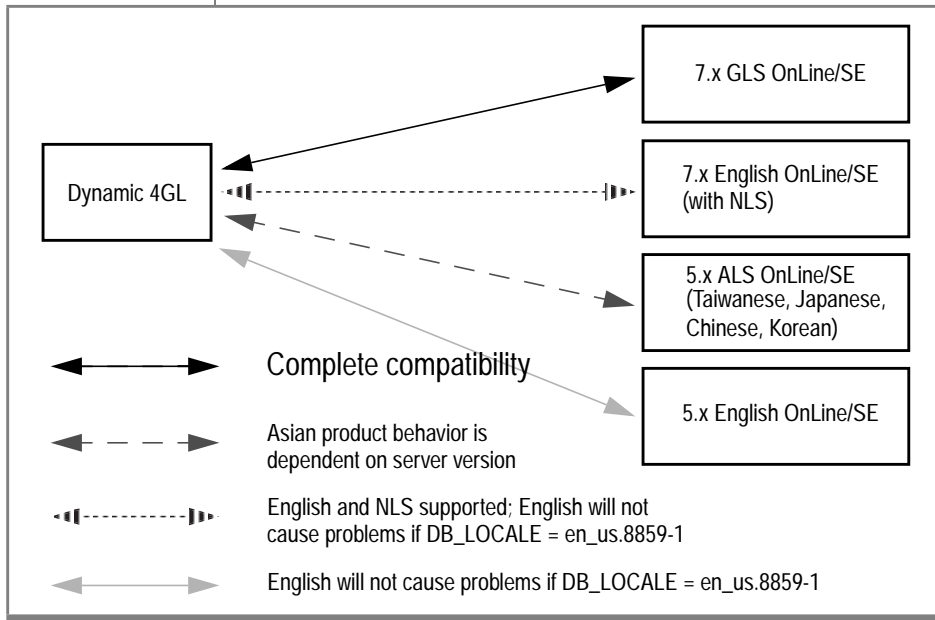


Figure D-1
Informix Database Servers that Dynamic 4GL Supports

Informix 7.2 and later GLS servers can store and retrieve data values that are compliant with single-byte and multibyte locales. GLS functionality requires the GLS version of INFORMIX-NET PC.

Dynamic 4GL is also compatible with Informix 5.x and 7.1 database servers, which can be English or non-English based.

Dynamic 4GL also supports older (ALS-based) Informix servers. The functionality differences are server-version based; applications might behave differently when connected to different servers.

Restrictions on Dynamic 4GL GLS Capability

When using Dynamic 4GL, the following restrictions apply:

- GLS features must be compiled to P code. Programs compiled to C code cannot be localized using GLS.
- GLS features in Dynamic 4GL are restricted to locales that use left-to-right text processing.
- Dynamic 4GL supports the entry, storage, and display of multibyte characters in some East-Asian languages, such as Korean, Japanese, and Chinese. However, these GLS features require a localized version of Windows.
- Dynamic 4GL provides limited support for the Thai language through code set **th_th.thai620**, with Language Supplement TH 7.20, for non-composite Thai characters. (Dynamic 4GL does not support composite Thai characters.)

Creating Dynamic 4GL Applications with GLS Support

This section outlines the steps that are needed to create localized Dynamic 4GL applications:

1. Set up the development environment.
The system administration tools you use must belong to the database server. You can use a UNIX terminal or a local terminal-emulation program on Windows (provided that it supports the local code set).
2. Write the code.
Filenames (source and compiled) must contain only English characters.
3. Compile and debug the code.
The Dynamic 4GL compiler can compile and link the components of the application.
The **fglmsg** message compiler can compile non-English text strings so that runtime messages can be displayed in the local language. The user interface of this message compile is in English.
Any Windows help requires the Windows Help Compiler.

4. Deploy the code.

Deployment is relatively unrestricted. Applications that can be created through the steps outlined here are *localized* applications for a specific locale, and therefore are not *internationalized*. (That is, they should not be used in another locale that requires, for example, a different code set from that of the message files.)

Compiling a Dynamic 4GL Application with GLS

Dynamic 4GL applications with GLS support can only be compiled to P code.

Important: *You cannot compile an application to C code.*

Creating a Runner

To create a GLS runner, add the `-gls` flag when running the `fglmkrun` script. For example:

```
fglmkrun -sh esql -gls
```

By default, the runner will be created in the `$FGLDIR/bin/gls` directory and linked into the `$FGLDIR/bin` directory. You can specify a runner location and name with the `-o` flag.

If the application is to run with Informix 7.3 database servers, set the `fglmkrun` flag for Informix Esq/C 9.1x. The flag changes from `-d ix730` to `-d ix914`. Alternatively, you could set the Dynamic 4GL environment variable `FGLDBS` to `ix914`.

If you do not specify the `-gls` flag when running the `fglmkrun` script, a runner using only ASCII characters will be created in the `$FGLDIR/bin/ascii` directory and linked into the `$FGLDIR/bin` directory. For example:

```
fglmkrun -sh c4gl
```

To create a GLS-aware P-code runner on Windows NT, place the following entry in the Makefile:

```
USE_GLS = YES
```



Checking if a Runner with GLS Support was Created

You can check if a runner was created with GLS support using the -V flag. A runner with GLS support enabled will display:

```
$ fgldr -V
INFORMIX Dynamic 4GL Runner Version 3.00
Built June 30 1999 15:36:04
(c) 1989-1998 Four J's Development Tools
Language support library: GLS (INFORMIX-ESQL Version 9.16.UC2)
Database front end : INFORMIX-ESQL VERSION 9.14(a)
```

A runner without GLS support enabled and using only ASCII characters will display:

```
$ fgldr -V
INFORMIX Dynamic 4GL Runner Version 3.00
Built June 30 1999 15:36:04
(c) 1989-1998 Four J's Development Tools
Language support library: standard ASCII (ISO8859-1)
Database front end : INFORMIX-ESQL VERSION 9.14
```

Localizing Dynamic 4GL Messages

Dynamic 4GL displays messages differently, depending on whether the message was generated by SQL functions or not.

Messages Generated by SQL functions

For messages generated by internal calls to SQL functions, these messages are retrieved using the Informix **rgetmsg** () function.

Dynamic 4GL Messages

Dynamic 4GL messages are stored in the **\$FGLDIR/<lang_state>/<codeset-id>/all.msg** file. In addition, Dynamic 4GL creates a subset of messages files that are similar to **i4gcl**, **c4gl**, and **form4gl**. Informix messages and stores them in **\$FGLDIR**. Dynamic 4GL uses these message files during compilation.

fglprofile Localized Messages

Message strings in the **fglprofile** are replaced by a link to a message number. If no message is found that corresponds to this number, the help message label appears. You can replace the message label with any string.

Before:

```
key.help.text="help"
```

After:

```
key.help.text= [msg -6900 "help"]
```

Using the Forms Compiler

The **fglform** forms compiler can process form specifications that include non-English characters that are valid in the client locale. It can also produce compiled forms that can display characters from the client locale, and that can accept such characters in input from the user.

Using the Message Compiler

The **fglmsg** message compiler can compile messages that include non-English characters, so that runtime messages can be in the local language. The **ERR_GET()** function can display locale-dependent characters.

Fully-Supported Windows Toolkit (WTK) Character Sets

Windows Toolkit (WTK) is the Dynamic 4GL software that customizes the Tcl/Tk for GUIs on Windows. The following WTK character sets can be used with Dynamic 4GL and are fully supported by Microsoft.



Important: You should refer to the Dynamic 4GL Release Notes for any changes to the supported WTK character sets.

Code Set	Character Set Name or Alias	Microsoft Code Page ID
1252 west-europe cp1252	ANSI_CHARSET, Western, Windows-1252	1252
1250 east-europe cp1250	EE_CHARSET, Central european (Windows), Windows-1250, x-cp1250	1250
1251 pc-slavic cp1251	RUSSIAN_CHARSET, Cyrillic (Windows), Windows-1251, x-cp1251	1251
1253 pc-greek cp1253	GREEK_CHARSET, Greek (Windows), Windows-1253	1253
1254 pc-latin5 pc-turkish cp1254	TURKISH_CHARSET, Turkish (Windows), Windows-1254	1254
1255 pc-hebrew cp1255	HEBREW_CHARSET, Hebrew Windows-1255	1255
1256 pc-arabic cp1256	ARABIC_CHARSET, Arabic Windows-1256	1256
1257 pc-baltic cp1257	BALTIC_CHARSET, Baltic Windows-1257	1257
1258	VIETNAMESE_CHARSET, Vietnamese, Windows-1258	1258
874	THAI_CHARSET, Thai, Windows-874	874

(1 of 2)

Fully-Supported Windows Toolkit (WTK) Character Sets

Code Set	Character Set Name or Alias	Microsoft Code Page ID
1361	JOHAB_CHARSET, Johab (Korean), Windows-1361	1361
932 CCSID932 sjis-s pc-sjis cp932	SHIFTJIS_CHARSET, Japanese, Windows-932, shift_jis,x-sjis, ms_Kanji, cs, ShiftJIS IBM CCSID 932 Mixed including 1880 UDC	932
ksc KS5601 cp949 57356	HANGUL_CHARSET, Korean, Korean (Wansung) KS C-5601-1987 Windows-949	949
gb GB2312-80 cp936	GB2312_CHARSET, Windows-936 Chinese (People's Republic of China, Singapore), Simplified Chinese Microsoft Windows	936
big5 Big-5 cp950 57352	CHINESEBIG5_CHARSET, Windows-950, Traditional Chinese MS Windows Code Page 950, Chinese (Hong Kong SAR, China Taiwan)	950

(2 of 2)

Partially Supported WTK Character Sets

The following table lists character sets that Dynamic 4GL can use, but for which Windows does not provide support for all characters. If you use any of these character sets, you will find some characters are missing when you type the complete character set. When possible, use the fully supported WTK character sets.

Code Set	Character Set Name or Alias	Microsoft Code Page ID
819 ASCII C 8859-1 Latin-1	IBM CCSID 819, C locale, POSIX Locale iso-8859-1, us-ascii, standard ascii, latin1 ibm819, iso-ir-6,ANSI_X3	1252
912 8859-2 Latin-2	iso-8859-1,iso-ir-101, ibm912, IBM CCSID 912	1250
813 8859-7 Latin-Greek	IBM CCSID 813, iso-8859-7, ibm813	1253
916 8859-8 Latin-Hebrew	IBM CCSID 916, ibm916, iso-8859-8	1255
920 8859-9 Latin-5	IBM CCSID 920, ibm920, iso-8859-9	1254
1089 8859-6 Latin-Arabic iso-ir-127 ASMO-708	IBM CCSID 1089, iso-8859-6, iso-ir-127	1256
57390 8859-13 Latin-Baltic	iso-8859-13	1257

Setting the CLIENT_LOCALE Variable

Use the values in the table to correctly set the **CLIENT_LOCALE** variable. The variable is created using the following syntax.

```
CLIENT_LOCALE= language_territory.codeSet
```

For example:

```
fr_ca.1252
```

The Code Set column in the following table lists the synonyms that can be used in the **CLIENT_LOCALE** variable.

Important: For more information on setting the **CLIENT_LOCALE** variable, refer to the “*Informix Guide to GLS Functionality*.”



Default Character Set

The *Latin* reference to Windows code pages denotes what is also called the *Roman* alphabet in U.S. English. In any locale, Dynamic 4GL requires at least one font that supports the code set if the application needs to produce output to the screen or to a report.

The default value on a UNIX computer is iso8859-1. This character set is not fully supported. You should set **CLIENT_LOCALE** to en_us.1252 if you want complete support of this character set.

For example:

```
CLIENT_LOCALE=en_us.1252
unset DB_LOCALE
(default value)
```

```
CLIENT_LOCALE=cs_cz.cp1250
DB_LOCALE=cs_cz.8859-2
```

```
CLIENT_LOCALE=ja_jp.sjis-s
DB_LOCALE=ja_jp.ujis
```

```
CLIENT_LOCALE=ko_kr.ksc
DB_LOCALE=ko_kr.ksc
```

```
CLIENT_LOCALE=ja_jp.sjis-s
DB_LOCALE=ja_jp.unicode
```

Internationalization and Localization

The terms *internationalization* and *localization* are near antonyms, but they both describe activities that are critical for applications that will be deployed in more than one locale. The first term, *internationalization*, refers to the work of analysts and developers who must design and write code that is generalized for different cultural contexts. The second term, *localization*, refers to the work of developers and translators who must adapt an internationalized application to the specific needs of a given linguistic or cultural setting.

Internationalization is the process of making software applications easily adaptable to different cultural and language environments.

Internationalization features support non-ASCII characters in character string values, and adaptable number, time, and currency formats. Internationalization also implies the ability to switch runtime environments from one language to another. Internationalization removes the need to recompile source code for a specific natural language or cultural environment.

A fully-internationalized application can run in different cultural environments with minimal adjustments, in some instances by simply exchanging language-specific files and setting up the operating environment.

An internationalized application must support the use of extended ASCII code sets. The default environment for 4GL is based on the ASCII code set of 128 characters. Each of these encoded values (or *code points*) requires seven bits of a byte to store each of the values 0 through 127, representing the letters, digits, punctuation, and other logical characters of ASCII. Because each ASCII character can be stored within a single byte, ASCII is called a *single-byte* character set. All other character sets that 4GL can support must include ASCII as a subset.

An internationalized application should, at a minimum, be *8-bit clean*. A program, GUI, or operating system is referred to as “8-bit clean” if it allows the high-order bit of a character code to take on a value of 1. 4GL applications are 8-bit clean, and therefore support the use of extended ASCII character sets, such as Windows code pages or ISO 8859 character sets.

Localization is the process of translating and adapting an internationalized product to specific language and cultural environments.

Localization usually involves setting the appropriate number, time, and currency formats for the intended country, as well as creating a translation of the runtime user interface (including help and error messages, prompts, menus, and reports).

You can reduce the cost and effort of localization if the application is designed with international requirements in mind. This release of 4GL supports localization in several areas:

- Entry, display, and editing of non-English characters
- References to SQL identifiers containing non-English characters
- Collation of strings containing non-English symbols
- Non-English formats for number, currency, and time values

For basic GLS concepts and for details of how Informix database servers and the INFORMIX-ESQL/C product implement GLS, see the *Informix Guide to GLS Functionality*.

Global Language Support Terms

Global language support (GLS) refers to the set of features that makes it possible to develop user interfaces and other parts of an application so that they can use non-Roman alphabets, diacritical marks, and so on. In order to understand the requirements of GLS, you will need to become familiar with the terms described in this section.

Code Sets and Logical Characters

For a given language, the *code set* specifies a one-to-one correspondence between each logical element (called a *logical character*, or a *code point*) of the character set, and the bit patterns that uniquely encode that character. In U.S. English, for example, the ASCII characters constitute a code set.

Code sets are based on logical characters, independent of the font that a display device uses to represent a given character. The size or font in which 4GL displays a given character is determined by factors independent of the code set. (But if you select, for example, a font that includes no representation of the Chinese character for *star*, then only whitespace will be displayed for that character until you specify a font that supports it.)

Collation Order

Collation order is the sequence in which character strings are sorted. Database servers can support collation in either *code-set order* (the sequence of code points) or *localized order* (some other predefined sequence). See the *Informix Guide to GLS Functionality* for details of localized collation.

4GL supports only code-set order. The database server, rather than 4GL, must do the sorting if you require localized collation of data values in NCHAR or NVARCHAR columns of the database. (You can write collation functions, but 4GL relational operators always use the code-set order.)

Single-Byte and Multibyte Characters

Most alphabet-based languages, such as English, Greek, and Tagalog, require no more than the 256 different code points that a single byte can represent. This simplifies aspects of processing character data in those languages; for example, the number of bytes of storage that an ASCII character string requires has a linear relationship to the number of characters in the string.

In non-alphabetic languages, however, the number of different characters can be much greater than 256. Languages like Chinese, Japanese, and Korean include thousands of different characters, and typically require more than one byte to store a given logical character. Characters that occupy two or more bytes of storage are called *multibyte characters*.

Locales

For 4GL (and for Informix database servers and connectivity products), a *locale* is a set of files that specify the linguistic and cultural conventions that the user expects to see when the application runs. A locale can specify these:

- The name of the code set
- The collation order for character-string data
- Culture-specific display formats for other data types
- The correspondence between uppercase and lowercase letters
- Determination of which characters are printable and which are nonprintable

The *Informix Guide to GLS Functionality* provides details of formats for number, currency, and time values. If no locale is specified, then default values are for U.S. English, which is the **en_us.8859-1** locale on UNIX systems, or Windows code page 1252. For deployment, 4GL is also delivered with the locale **en_us.1252@dict**, which corresponds to that Windows code page.

The locale **en_us.1252@dict** allows you to compile and run programs that contain non-English characters from any single-byte language, but the default data formats are those of U.S. English. Alternatively, you can use the **Setnet32** utility to specify some nondefault locale, such as one of those listed in [“Locales Supported by 4GL” on page D-20](#).

Global Language Support

GLS is a set of features that enable you to create localized applications for languages other than U.S. English and for country-specific cultural issues, including the localized representation of dates, currency values, and numbers. 4GL supports the entry, retrieval, and display of multibyte characters in some East Asian languages, such as Japanese and Chinese.

The following GLS-enabled built-in functions or operators have been modified since the 6.0 release of 4GL to provide support for non-English locales. Some can accept multibyte characters as arguments or operands, or can return values that include multibyte characters.

- CLIPPED operator
- DOWNSHIFT()
- FGL_GETENV()
- FGL_KEYVAL()
- LENGTH()
- Substring ([]) operator
- UPSHIFT()
- WORDWRAP operator

See the *INFORMIX-4GL Reference Manual* for the syntax and semantics of these built-in functions and operators. (Besides these, certain other built-in functions and operators of 4GL can also process or return multibyte values.)

Native Language Support

The GLS capability of 4GL is not a logical superset of native language support (NLS) as that term is used by Informix. An Informix NLS server is one that recognizes the NCHAR and NVARCHAR data types. Such servers can communicate with client applications in single-byte locales.

4GL supports Informix NLS servers at the *implicit* level of compliance, through INFORMIX-NET and through INFORMIX-ESQL/C. The 4GL language does not recognize NCHAR or NVARCHAR data types, but such values from the database server are automatically converted to CHAR and VARCHAR values, and 4GL can use CHAR and VARCHAR values to update NCHAR and NVARCHAR columns, provided that an operating-system locale exists in the GLS directory for the NLS locale.

The **DBNLS** value that is set on the client system running 4GL is passed to the database server, but any **LC_COLLATE** value from the client is ignored. (Collation by the 4GL application is based on the code-set order, not on **LC_COLLATE**, but the database server can perform localized collation of NCHAR or NVARCHAR column values, based on the **LC_COLLATE** setting.)

The **COLLCHAR** environment variable is not required to enable NLS; on the contrary, 4GL requires that **COLLCHAR** not be set. If you have **COLLCHAR** set to 1, you must reset it to NULL. For more information about **DBNLS**, **COLLCHAR**, and **LC_COLLATE**, see the *Informix Guide to GLS Functionality*.

Non-GLS Components of This Release

Not all components of 4GL provide GLS. This section identifies components of 4GL that support only single-byte locales that do not require bidirectional text processing.

The absence of GLS does not imply that these features are unavailable in non-English locales. It does imply, however, a restriction to locales that require only single-byte code sets and left-to-right text processing.

Installation in Non-English Locales

This section identifies the general requirements for installation of 4GL in non-English locales. Because *non-English* refers to all locales other than **en_us.8859-1** (for UNIX) or **en_us.1252@dict** (for Windows), most locales of the English-speaking world are non-English in this context, as are the locales of most of the rest of the world.

The directory structure of Informix GLS products is shown in [Figure D-2](#).

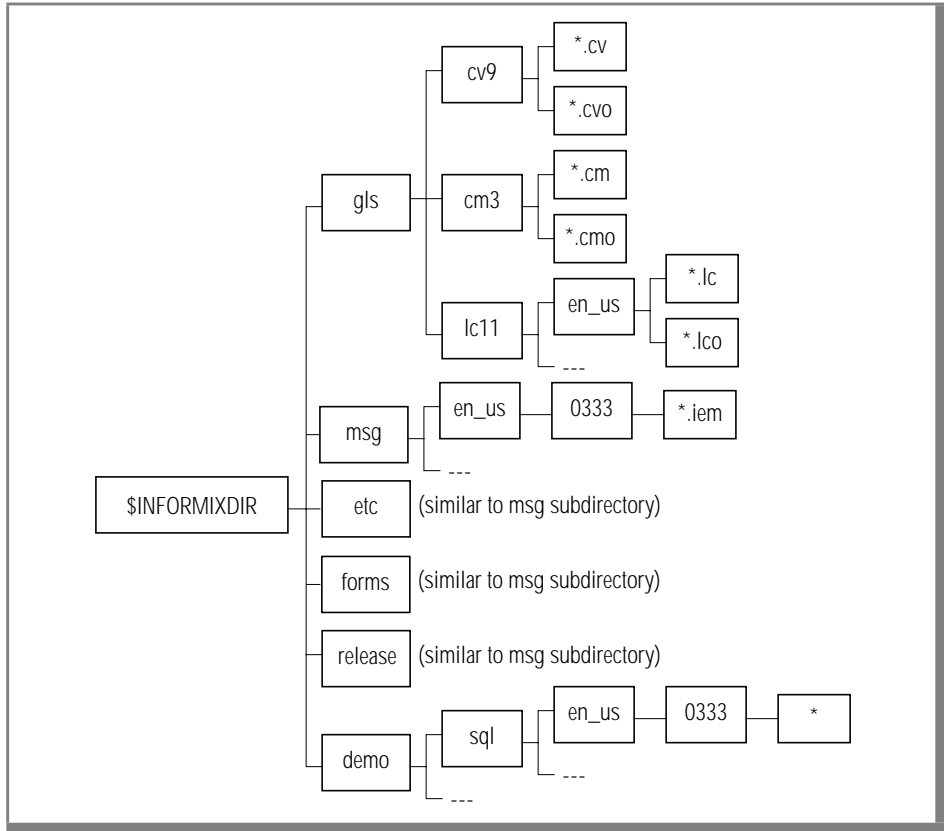


Figure D-2
Directory Structure
of GLS Products

Requirements for International Application Development

The following requirements must be met to develop a 4GL application that is fully adapted to a language or to a country:

- The targeted hardware platform and operating system need to support the desired language and country combination.
The operating-system environment on both the client platform and the server platform might require special versions to support the entry, manipulation, and display of non-English data.
- The Informix products need to support the language. Informix products are 8-bit clean and allow entry, manipulation, and display of most European and Asian language data.
- Error messages generated by 4GL and the database server should be available in a localized version, so that only local languages appear in the runtime environment.
- All parts of the user interface created by the application developer (such as menus, prompts, error messages, and help) should be translated into the target language.

In many cases, the last three of these four requirements can be met by using an Informix language supplement. Your Informix sales representative can advise you regarding the availability of language supplements, of localized versions of Windows, and of database servers that are compatible with 4GL.

Language Supplements

Use of 4GL with some non-English languages might require an Informix *language supplement* specific to the conventions of the country or language. Language supplements are currently required, for example, for Informix database servers to support each of the following East Asian languages.

Country or Language	Informix Language Supplement
People's Republic of China	Language Supplement ZHCN 7.20
Taiwanese	Language Supplement ZHTW 7.20

Country or Language	Informix Language Supplement
Japanese	Language Supplement JA 7.20
Korean	Language Supplement KO 7.20
Thai (simplified)	Language Supplement TH 7.20

Language supplements for these East Asian languages include locale files, translated message files, and translated menu files. Localized versions of 4GL for East Asian locales (for example, Japanese 4GL) will include the relevant files. See the release notes for additional information.

A corresponding International Language Supplement includes locale files and code-set conversion files for most European languages. Because most of these files are included with the INFORMIX-NET (7.2) connectivity software that is provided with 4GL, this supplement need not be purchased by 4GL customers unless the required locale is not included with 4GL.

When the Informix database server is installed in locales based on non-English European languages, both the default (English) database server and the International Language Supplement must be installed.

When 4GL is installed, the locale files must also be installed. Contact your Informix sales office for information regarding current support for specific locales.

Locales Supported by 4GL

A *locale* is the part of the processing environment that defines conventions for a given language or culture, such as formatting time and money values, and classifying, converting, and collating characters. The Informix GLS locale definition is similar to the X/Open CAE Specification.

Code sets that WTK 4GL supports include those listed in the following table.

Country or Language	Windows Code Page
People's Republic of China	936 (also known as GB 2312-80)
Taiwanese	950 (also known as Big-5)
Japanese	932 (also known as Shift-JIS)
Korean	949 (also known as KSC 5601)
Eastern European (Latin)	1250
Eastern European (Cyrillic)	1251
Western European (Latin)	1252
Greek	1253
Turkish	1254

Here *Latin* in reference to Windows code pages 1250 and 1252 denotes what is also called the *Roman* alphabet in U.S. English. In any locale, 4GL requires at least one font that supports the code set, if the application needs to produce output to the screen or to a report.

4GL provides limited support for the Thai language through code set **th_th.thai620**, with Language Supplement TH 7.20, for non-composite Thai characters. (4GL does not support composite Thai characters.)

Client Locales and Server Locales

The locale of the system on which the 4GL application is running is called the *client locale*. For an application that is partitioned through 4GL, this refers to the locale of the application server and of the display server. The locale of the database server is called the *server locale*. [“Handling Code-Set Conversion” on page D-49](#) describes special procedures that might be required if the client locale and the server locale are not identical.

Setting Environment Variables for Specific Locales

4GL requires that environment variables be set correctly on UNIX systems that support the database server or 4GL applications that support application server and display server partitions. For details about setting environment variables on UNIX systems for GLS, see the *Informix Guide to GLS Functionality*. See also “[Configuring the Language Environment](#)” on [page D-37](#) of for additional information about setting environment variables.

To set environment variables on Windows 95 systems, you set most environment variables in the Windows 95 or NT 4.0 registry by using the **Setnet32** utility.

See the *Informix Guide to GLS Functionality* for an example of non-English locale files.

Requirements for All Locales

This section outlines the steps that are needed to create localized 4GL applications:

1. Set up the development environment.

The system administration tools that you use must belong to the database server. You can use a UNIX terminal or a local terminal-emulation program on Windows (provided that it supports the local code set).

2. Write the code.

Filenames (source and compiled) must contain only English characters.

3. Compile and debug the code.

The 4GL compiler can compile and link the components of the application.

The Message Compiler can compile non-English text, so that runtime messages can be displayed in the local language. The user interface of the Message Compiler is in English.

Any Windows help requires the Windows Help Compiler.

The INFORMIX-4GL Interactive Debugger is not GLS-enabled. (The Dynamic 4GL Debugger has sufficient GLS capability to display non-English characters from the client locale.)

4. Deploy the code.

Deployment is relatively unrestricted. Applications that can be created through the steps outlined here are *localized* applications for a specific locale, and therefore are not *internationalized*. (That is, they should not be used in another locale that requires, for example, a different code set from that of the message files.)

The 4GL Compilers

The compilers have limited GLS capability, as the sections that follow describe.

The 4GL Character Set

4GL keywords, identifiers, delimiters, and special symbols in source code are restricted to the same ASCII characters described in the *INFORMIX-4GL Reference Manual*. Additional printable characters from the client locale, however, are valid within source code files in the following contexts only:

- Within comments
- Within 4GL identifiers
- Within certain SQL identifiers (as listed in the table in [“SQL and 4GL Identifiers”](#) on page D-25)
- Within expressions where character-string literals are valid

In non-English locales, 4GL identifiers can include non-ASCII characters in identifiers if those characters are defined in the code set of the locale that `CLIENT_LOCALE` specifies. In multibyte East Asian locales that support languages whose written form is not alphabet-based, a 4GL identifier need not begin with a letter, but the storage length cannot exceed 50 bytes. (A Chinese identifier, for example, that contains 50 logical characters would exceed this limit if any logical character in the identifier required more than one byte of storage.)

Non-English characters in other contexts, or characters that the client locale does not support, will generally cause compilation errors.

At runtime, the user can enter, edit, and display valid characters from the code set of the client locale. Whether a given character from a non-English code set is *printable* or *nonprintable* depends on the client locale.

Values that include non-English characters can be passed between a 4GL application and the database server, if the client and server systems have the same locale. If the locales are different, data can still be transferred between the 4GL client and the database server, provided that the client locale includes appropriate code-set conversion tables. See “[Configuring the Language Environment](#)” on page D-37 or the *Informix Guide to GLS Functionality* for information about establishing a locale and about code-set conversion between locales. See also “[Handling Code-Set Conversion](#)” on page D-49.

Non-English Characters

The following features of the 4GL compiler are GLS-enabled to support non-English characters that are valid in the client locale:

- Names of identifiers
- Values of CHAR and VARCHAR variables and formal arguments
- Characters within TEXT blobs
- Message text, quoted strings, and values returned by functions
- Text within comments, forms, menus, and output from reports

Named 4GL program entities include variables, functions, cursors, formal arguments, labels, reports, and prepared objects. 4GL has a limit of 50 bytes on the lengths of these names, but C compiler or linker restrictions might impose lower limits.

SQL and 4GL Identifiers

SQL identifiers are the names of database entities, such as table and column names, indexes, and constraints. The first character must be an alphabetic character, as defined by the locale, or an underscore (= ASCII 95) symbol. You can use alphanumeric characters and underscores (_) for the rest of the SQL identifier. Most SQL identifiers can be up to 18 bytes in length. What characters are valid in SQL identifiers depends on the locale of the database server. Neither single-byte nor multibyte whitespace characters can appear in SQL identifiers.

SE

For INFORMIX-SE database servers, whether non-English characters are permitted in the names of databases, tables, or log files depends on whether the operating system permits such characters in filenames. ♦

The user interface of the 4GL compiler is in English. If edit fields contain multibyte characters, there is no checking, and the results might be unpredictable. Embedded SQL statements can include valid non-English identifiers for some database entities. The following tables summarize the instances where non-English characters are valid as identifiers within 4GL source code modules. The first table lists SQL identifiers.

SQL Identifier	Allow Non-English Characters?
Column name	Yes
Constraint name	Yes
Database name	Yes (Operating System limitations on INFORMIX-SE)
Index name	Yes
Log filename	Yes (Operating System limitations on INFORMIX-SE)
Stored procedure name	Yes
Synonym	Yes
Table name	Yes (Operating System limitations on INFORMIX-SE)
View name	Yes

The following 4GL identifiers allow non-English characters.

4GL Identifier	Allow Non-English Characters?
Variable name	Yes
Cursor name	Yes
Filename or pathname	No
Formal argument name	Yes
Function or report name	Yes
Prepared statement name	Yes
Statement label	Yes

Input and output filenames for the 4GL compiler cannot be localized. Only ASCII characters are valid in input and output pathnames or filenames. (If support for uppercase ASCII letters is required, specify **en_us.1252@dict** as the locale at compile time. Uppercase letters are *not* defined in **en_us.1252**.)

Collation Sequence

The collation (sorting) sequence in 4GL statements is implied by the *code-set order* in the files that define the client locale. (Any collating that is specified by the **LC_COLLATE** value of the client locale is ignored.) Collation in SQL operations (where the database server uses its own collation sequence) depends on the data type and on the server locale (which can specify a localized order of collation). It is possible for the 4GL application and the database server to use a different collating sequence, or for a 4GL application to connect to two or more servers that use different collating sequences. The collation sequence can affect the value of Boolean expressions that use relational operators and the sorted order of rows in queries and in reports.

Locale Restrictions

The compiler requires the **en_us.0333** locale. It accepts as input any source file containing data values in the format of the client locale. The compiler can generate binaries or P-code files with client-locale text strings. The runtime locale of a 4GL program must be the same as its compile-time locale.

As a convenience to the developer, 4GL adds a field in P-code header files to specify the locale in which the files were compiled but does not compare these locales.

The Forms Compiler

The **fglform** forms compiler can process form specifications that include non-English characters that are valid in the client locale. It can also produce compiled forms that can display characters from the client locale, and that can accept such characters in input from the user.

The Message Compiler

The **mkmessage** message compiler has a user interface in English but can compile non-English text into runtime messages in the local language.

On-Line Help

Help for 4GL applications in non-English locales requires the native Windows Help facility.

East Asian Language Support

4GL can create applications for Asian languages that use multibyte code sets. This support is *only* available when 4GL applications are developed and run under a multibyte version of Microsoft Windows or UNIX.

4GL supports the following features in multibyte locales:

- Menu items, identifiers, and text labels in the native language
- Features to avoid the creation of partial characters
- Non-English data within 4GL applications
- Cultural conventions, including the representation of date, time, currency, numeric values, and localized collation
- Kinsoku processing for Japanese language text with WORDWRAP
- Icon modification without changing the 4GL application binary
- Text geometry that adjusts automatically to meet localization needs
- Application comparisons that adopt the comparison rules and collating sequence that the locale defines implicitly (SQL comparison and collation depend on the database server.)

This version of 4GL does not support composite characters, such as are required in code sets that support the Thai language.

4GL comments and character string values can include multibyte characters that are supported by the client locale in contexts like these:

- Character expressions and multiple-value character expressions
- Literal values within quoted strings
- Variables, formal arguments, and returned values of CHAR, VARCHAR, and TEXT data types

Multibyte characters can also appear in 4GL source code (or in user-defined query criteria) that specifies the SQL identifier of any of the database objects listed in the table on “[SQL and 4GL Identifiers](#)” on page D-25. 4GL does not, however, support multibyte characters as currency symbols or as separators in display formats that **DBDATE** or **DBFORMAT** specifies.

Logical Characters

Within a single-byte locale, every character of data within character-string values requires only a single byte of memory storage, and a single character position for display by a character-mode device.

This simple one-to-one relationship in character-string operations between data characters, display width, and storage requirements does not exist in East Asian locales that support multibyte characters. In such locales, a single logical character might correspond to a single byte or to two or more bytes. In such locales, it becomes necessary to distinguish among the *logical characters* within a string, the *display width* that the corresponding glyph occupies in a display or in report output, and the number of *bytes* of memory storage that must be allocated to hold the string.

In locales that support multibyte characters, some built-in functions and operators that process string values operate on logical characters, rather than on bytes. For code sets that use multibyte characters, this modifies the byte-based behavior of several features in 4GL. A single logical character can occupy one or more character positions in a screen display or in output of a report, and requires at least one byte of storage, and possibly more than one.

Declaring the CHAR or VARCHAR data types of variables, formal arguments, and returned values is *byte*-based. Runtime processing of some character strings, however, is done on a *logical character* basis in multibyte locales.

Partial Characters

The most important motivation for distinguishing between logical characters and their component bytes is the need to avoid partial characters. These are fragments of multibyte characters. Entering partial characters into the database implies corruption of the database, and risks malfunction of the database server.

Partial characters are created when a multibyte character is truncated or split up in such a manner that the original sequence of bytes is not retained. Partial characters can be created during operations like the following:

- Substring operations
- INSERT and UPDATE operations of SQL
- Word wrapping in reports and screen displays
- Buffer to buffer copy

4GL does not allow partial characters and handles them as follows:

- Replaces truncated multibyte characters by single-byte whitespaces
- Wraps words in a way that ensures that no partial characters are created in reports and screen displays
- Performs code-set conversion in a way that ensures that no partial characters are created

For example, suppose that the following SELECT statement of SQL:

```
SELECT col1[3,5] FROM tab1
```

retrieved three data values from **col1** (where **col1** is a CHAR, NCHAR, NVARCHAR, or VARCHAR column); here the first line is not a data value but indicates the alignment of bytes within the substrings:

AA ² BB ² AA	becomes	"s ¹ Bs ¹ "
ABA ² C ² AA	becomes	"A ² s ¹ "
A ² B ² CABC	becomes	"B ² C"

Here the notation s¹ denotes a single-byte whitespace. Any uppercase letter followed by a superscript (²) means an East Asian character with multibyte storage width; for simplicity, this example assumes a 2-byte storage requirement for the multibyte characters. In the first example, the A² would become a partial character in the substring, so it is replaced by a single-byte whitespace. In the same substring, the B² would lose its trailing byte, so a similar replacement takes place.

General Guidelines

This section lists the issues that you need to consider when writing and translating applications.

Internationalization Guidelines

To make a 4GL application world-ready, keep the following guidelines in mind:

- Do not assume that application users are English-speaking or will accept any pre-set business rules or formats.
- Use code libraries wherever possible. This centralizes common code and makes changes and maintenance easier when developing for international markets.

Specific programming areas that might require special attention (and that are treated in detail in the *Informix Guide to GLS Functionality*) include:

- character-string display, entry, storage, retrieval, and processing.
- formats for literal date, time, currency, and numeric values.
- code-set conversion between client and server.
- In all windows that will appear in more than one language, consider differences in word length among languages when you are designing the window and its graphical objects.
- Allow space for the expansion of user message strings. Brief English strings such as *Popup* can double in size as a result of translation. On average, you can expect a 30 percent increase in the size of messages.
- When designing windows, remember that names, addresses, dates, times, and telephone numbers have different formats in different countries.
- When possible, use picture buttons instead of buttons with titles.
- Consider that measurement systems can also differ. Most countries outside the U.S. express quantities using the metric system. For example, liters, centimeters, and kilometers instead of quarts, inches, and miles.

- Make sure that all screens, menus, user messages, reports, help facilities, and application parameters (such as holidays, bank years, formulas) that were developed with Informix tools for the application are either table-driven or are controlled by text files or environment variables that are easy to modify. This issue is discussed later in this appendix.
- Avoid embedding any messages, prompts, or elements of the user interface into the source code of the program. Ideally, all user interface elements can be switched dynamically by referencing a different set of translated files.
- Consider different keyboard layouts. A character (such as “/”) that is easily accessible on an ASCII keyboard might require several keystrokes in the standard keyboard of some other country.
- Consider creating a configuration utility to deal with different font types. Some applications that will be deployed in several different countries might need to load different fonts to accommodate specific national characters.

Because these fonts are often supplied by third parties, you might not be able to predict the font names when you develop the application. In this case, you can use the default font names and provide a configuration utility that allows the user to specify the font name before running the application.

- Consider differences in paper size when designing reports. Most countries outside the U.S. use the ISO Standard A4 paper size, which is 21 by 29.7 centimeters, slightly longer and narrower than the American standard 8.5 by 11 inches.
- Avoid fragmentation of messages or potentially ambiguous key or command words. Avoid determining variable portions of a message at runtime; for example, the differing syntax of other languages can make the order in which your functions return parameters an obstacle to correct translation.
- Wherever possible, avoid abbreviations, acronyms, contractions, and slang.
- Place comments around any string pertaining to the user interface to facilitate localization.

- Use localized error messages and help files. The message compiler utility that is provided with 4GL enables you to create customized help files as well as a localized version of the 4GL runtime message file. (This is the **4glusr.msg** file in the **msg** directory.) Internationalizing messages is further discussed in “[Localizing Prompts and Messages](#)” on page D-47.
- You can handle reports (which are 4GL programs) in the same way that you internationalize the rest of your 4GL source code.

If your database server and ESQ/C API are Version 6.0 or later, you might be able to take advantage of Native Language Support (NLS) functionality, even though 4GL provides only implicit support for NLS. For more information, see the *Informix Guide to SQL: Reference*.

Localization Guidelines

Localization refers to the actual process of adapting the application to the cultural environment of end users. This process often involves translation of the user interface and user documentation and can be quite time consuming and costly. Here are some guidelines to follow:

- Consult the native operating-system internationalization guide. Most platforms provide documentation on internationalization. This material might help you determine which date, time, and money formats are appropriate for the target language and culture. For more information about internationalization and Windows, see “International Applications” in the *Microsoft Windows Programmer’s*. For more information about internationalizing Informix products in the UNIX environment, see the *Informix Guide to SQL: Reference*. For information about the terms and constructs of GLS technology, see the *Informix Guide to GLS Functionality*.
- Make sure the targeted hardware, operating-system environments, and Informix product versions of your applications can support the desired language and culture.

- Find out if the runtime environment of 4GL and of the database server is currently available in the target language.

For example, the 4GL runtime environment (and the Informix Dynamic Server administrator's environment) is usually translated into several languages, including French, German, Spanish, Russian, and Japanese.

- Keep a glossary of all strings and keywords in a database or text file.

This glossary will make it easier to see which messages are duplicated throughout the source code. The glossary will also increase the consistency of terms and language in the user interface throughout the application. Once the glossary is created for one language, it can be used for product updates and additional localizations.

- Create a mechanism that allows a glossary to drive the definition of the user interface.

This can be particularly useful if you expect to localize the application often. A translator can edit the glossary without having to understand the source code of the application. Your tool can then create the user interface from the translated glossary, and the translator can focus on making cosmetic enhancements to the translation (such as positioning the messages appropriately) and correcting minor errors.

- Consider creating a checklist of those user interface elements in your application that should be externalized into text files from the source code, and therefore from the compiled portion of the program. These text files can then be modified even after the program is compiled. Externalize the following elements:

- Menus
- Forms
- Messages
- Labels
- Help (.msg) text
- Numeric, date, time, and money formats
- Report names

- Consider retaining a professional translator for some or all of this process.

A faulty translation is costly. You can spend a great deal of time and money correcting errors in your localized product. And if you do not correct the problems, your users will be dissatisfied with your application.

Localization Methodology Overview

This section lists the elements of an application and indicates some ways in which each can be localized. This overview, while not comprehensive, illustrates how to approach a project of this nature. The rest of this appendix expands on the approaches listed here.

For many of the application elements discussed in this section, the two methods of localization are the *table-based* approach and the *file-based* approach. The table-based approach involves obtaining translation information from a database using SQL queries. The file-based approach involves retrieving the values of the variables from a text file.

Application Help and Error Messages

The following methods are available for localizing application help and error messages.

Table-Based Localization of Messages

To use this method, you need to verify the availability of tables. It often also requires the hard coding of defaults in case the database cannot be accessed.

File-Based Localization of Messages

This method uses the **fglmkmessage** message compiler utility to create help and error message files. For more information, see [“Localizing Prompts and Messages” on page D-47](#).

Date, Time, and Currency Formats

To localize formats for dates, time, and money values, set the Informix environment variables **DBDATE**, **DBFORMAT**, and **DBMONEY**. Formatting conventions of some East Asian locales require that the **GL_DATE** or **GL_DATETIME** environment variable be set.

Informix System Error Messages

The following methods are available for localizing Informix system messages and error messages.

Informix Translation

Informix provides error message translation for a variety of languages. You can use the **DBLANG** environment variable to point to a message directory containing translated messages. Contact your local Informix sales office for a list of available language translations.

Customized System Error Message Files

If no Informix translation of the error messages is available, and if the source code of error message files is delivered with the product, you can localize the message source files using the **fglmessage** utility. For more information, see [“Localizing Prompts and Messages” on page D-47](#).

Code-Set Conversion

The method available depends on whether you are using UNIX or Windows:

- For UNIX systems, set the **DBAPICODE** environment variable.
- For Windows systems that use INFORMIX-NET with Dynamic 4GL, set the **CLIENT_LOCALE** and **DB_LOCALE** entries in the registry.

For details, see [“Handling Code-Set Conversion” on page D-49](#).

Configuring the Language Environment

Environment settings that affect the language environment exist both in your 4GL environment and in your system environment. Using the GLS features of 4GL with Informix database servers involves several compatibility issues:

- The English servers create English databases with ASCII data. For these, the 4GL program must access the servers with **DB_LOCALE** set to **en_us.8859-1**.
- The 5.x ALS versions of Informix servers can use variables such as **DBCODESET** and **DBCSOEVERIDE** as substitutes for **DB_LOCALE** and **DBCONNECT**, respectively. These environment variables need to be set by using **Setnet32**.
- The 5.x ALS versions use **DBASCIIBC** to emulate the 4.x ASCII servers. This environment variable should be set in the registry, if such behavior is desired.
- The **SERVER_LOCALE** environment variable is set on the database server, not on the 4GL client. This specifies the locale that the database server uses to read or write operating-system files. If this is not set, the default is U.S. English (**en_us.8859-1**).

If no setting is specified, the 4GL application uses an English locale. But the registry sets everything to the local language, code set, or locale, so the practical default is for applications to use the local locale.

The non-internationalized portions of the product are initialized with the default (U.S. English) locale. That is, both **CLIENT_LOCALE** (**en_us.1252**) and **DB_LOCALE** (**en_us.8859-1**) are set to English. This initialization is necessary because many common functions are shared between the internationalized and non-internationalized components.

Important: Except for **DBFORMAT**, all the environment variables that are described in the sections that follow apply to Informix database servers.



Consider also the following points:

- The application cannot support connections to different databases with different locales concurrently; for example, in extended joins.
- The environment variables discussed here deal with the environment **DB_LOCALE** that is passed to the server.
- **CLIENT_LOCALE** cannot be changed dynamically during execution.
- The previous point has one exception: the **CLIENT_LOCALE** can always be set to English (because English is a subset of all locales).

When connecting to a GLS, NLS, or ALS (Asian Language Support) database, the **DB_LOCALE** code set should match the **DB_LOCALE** code set of the database. Otherwise, data corruption can occur, because no validation of code-set compatibility is performed by the server. An ALS server can refuse the connection when the code sets do not match, but an NLS server cannot.

Environment Variables That Support GLS

This section examines the environment variables that support the GLS capabilities of 4GL, including the following 4GL environment variables:

- **DBDATE** defines date display formats.
- **DBMONEY** defines monetary display formats.
- **DBFORMAT** defines numeric and monetary display formats and has more options than **DBMONEY**.

INFORMIX-4GL also supports the following GLS environment variables:

- **DB_LOCALE** is the locale of the database to which the application is connected.
- **CLIENT_LOCALE** is the locale of the system that is executing the 4GL application.
- **DBLANG** points to the directory for language-specific message files that an Informix product uses, such as Informix error messages.
- **GL_DATE** defines date displays, including East Asian formats.
- **GL_DATETIME** defines date and time displays, including East Asian formats.
- **SERVER_LOCALE** is the locale of the database server for file I/O.

4GL does not use **DB_LOCALE** directly; this variable, as well as **DBLANG**, is used by the GLS version of INFORMIX-NET PC. See the *Informix Guide to GLS Functionality* for details on how **DBLANG**, **DB_LOCALE**, **GL_DATE**, and **GL_DATETIME** are set.

Compatibility Issues

In order for 4GL to work with older Informix database servers (such as 5.x ALS), it is necessary for these environment variables to be set in the Windows registry. This is done by the GLS version of INFORMIX-NET PC. When the 4GL application accesses an NLS database, appropriate NLS environment variables must be set in the registry if NLS functionality is desired.

DBAPICODE

This environment variable specifies the name of a mapping file for peripheral devices (for example, a keyboard, a display terminal, or a printer) whose character set is different from that of the database server.

DB_LOCALE

This environment variable specifies the locale of the database to which the 4GL component or application is connected. The only Informix databases that currently support non-English languages exist in UNIX. Therefore, when the locales are non-English, the localized 4GL application can only connect to these databases. The format for setting **DB_LOCALE** is **DB_LOCALE=<locale>**.

Consider also the following points regarding **DB_LOCALE**:

- If the application uses this value to access a database, the locale of that database must match the value specified in **DB_LOCALE**. If it does not match, the database connection might be refused (unless **DBCNOVERRIDE** is set to 1), depending on the server version.
- If a database is created, then this new database has the value specified by **DB_LOCALE**.
- If **DB_LOCALE** is invalid, either because of wrong formatting or specifying a locale that does not exist, then an error is issued.

- If the code set implied by **DB_LOCALE** cannot be converted to what **CLIENT_LOCALE** implies, or vice versa, an error is issued.
- If **DB_LOCALE** is not specified, there is no default value; in this case, the GLS version of INFORMIX-NET PC behaves as if code-set conversion were not needed.

CLIENT_LOCALE

This environment variable specifies the locale of the (input) source code and the compiled code (to be generated). This is also the locale of the error files (if any) and the intermediate files. The format of **CLIENT_LOCALE** is the same as that of **DB_LOCALE**:

- The characters that reach the user interface (the non-ASCII characters) must be in the **CLIENT_LOCALE**.
- If **DB_LOCALE** is invalid, either because of wrong formatting or specifying a locale that does not exist, an error is issued.
- The **DB_LOCALE** and **CLIENT_LOCALE** settings need to be compatible, meaning there should be proper code-set conversion tables between them. Otherwise, an error is issued.
- If **CLIENT_LOCALE** is not set in the Windows client, then Windows code page 1252 is the default.
- The **CLIENT_LOCALE** must match the environment of the user interface (meaning that it should be compatible with the local version of Windows). Otherwise, an error is issued.
- Collation by the 4GL application follows the code-set order of **CLIENT_LOCALE**, except in SQL statements (where the database server uses its own collation sequence). Any **LC_COLLATE** specification is ignored.

DBLANG

The value of **DBLANG** is used to complete the pathname to the directories that contain the required message, help, and demo files. The format of **DBLANG** is the same as that of **DB_LOCALE**:

- If **DBLANG** is not set, the value defaults to that of **CLIENT_LOCALE**.
- If **DBLANG** is invalid, then **en_us.1252** is the default value. This case occurs if **DBLANG** is improperly formatted, or if it points to a locale that does not exist, or points to a locale that is incompatible with the version of Windows on which the 4GL application is running.

See also the description of **DBLANG** in the *Informix Guide to GLS Functionality*.

DBDATE

The **DBDATE** environment variable has been modified to support era-based dates (Japanese and Taiwanese). The days of the week and months of the year (in local form) are stored in the locale files. If this environment variable is set, it might override other means of specifying date formats.

DBMONEY

This environment variable has been modified to accept multibyte currency symbols. 4GL components that read the value of **DBMONEY** (or **DBFORMAT**) must be able to correctly process multibyte characters as currency symbols. If **DBMONEY** is set, its value might override other means of specifying currency formats.

DBFORMAT

This environment variable has been modified to accept multibyte currency symbols. Unlike the version of **DBFORMAT** for English products, display of the decimal point is optional, rather than mandatory, in 4GL. (Use of a comma as the **DBFORMAT** decimal separator can produce errors or unpredictable results in SQL statements in which 4GL variables are expanded to number values that are formatted with commas as the decimal separator.)

If **DBFORMAT** is set, its value can override other means of specifying number or monetary formats.

The **gfiles** utility is described in the *Informix Guide to GLS Functionality* and is packaged with INFORMIX-4GL and INFORMIX-SQL products. This utility allows you to generate lists of the following files:

- GLS locales available in the system
- Informix code-set conversion files available
- Informix code-set files available

Default Values of GLS Environment Settings

Default values assumed by INFORMIX-4GL and INFORMIX-SQL products (which differ from those of ALS environments) are described in this section.

The following table shows the values assumed by 4GL when you define only some of the required values of locales.

(A value of `ja-jp.ujis` is assumed in the following example. CL means **CLIENT_LOCALE**, and DL means **DB_LOCALE**.)

User Defined				Values in Product	
CL Defined	CL Value	DL Defined	DL Value	CL Value	DL Value
No	--	No	--	en_us.8859	en_us.8859
Yes	ja_jp.ujis	No	--	ja_jp.ujis	ja_jp.ujis
Yes	ja_jp.ujis	Yes	ja_jp.ujis	ja_jp.ujis	ja_jp.ujis
No	--	Yes	ja_jp.ujis	en_us.8859	ja_jp.ujis

If you do not set the **DBLANG** environment variable, it is set to the value of **CLIENT_LOCALE**.

System Environment Variables

This section describes how you can query your system environment for language and country variables.

Windows Environment Variables

To access the language environment variable programmatically, you can use any of the following three approaches:

- Read the **Language** value directly from the [ResourceLocale] section of the registry, specifically:

```
HKEY_CURRENT_USER\Control Panel\Desktop\ResourceLocale
```

This has a 32-bit numeric value called a locale ID, part of which defines the language that is used in the Windows user interface. The numbers are defined in **WINNT.H**.

For example, if `ResourceLocale` is set to 00000409, the 10 lower-order bits 000001001 (= hexadecimal 009) represent English, the constant `LANG_ENGLISH` in the `winnt.h` file. If the value is set to 00000401, then 001 represents Arabic (`LANG_ARABIC`).

(You can also read and modify this setting through the Control Panel.)

On Windows 95 and Windows NT, use the registry, rather than **.ini** files, but **.ini** files are supported (so that you can install pre-Windows 95 programs).

Some of the language codes that Windows currently supports are listed in the following table.

Code	Language	Code	Language
ENU	U.S. English	FRC	Canadian French
ENG	U.K. English	ISL	Icelandic
DAN	Danish	ITA	Italian
DEU	German	NLD	Dutch
ESN	Modern Spanish	NOR	Norwegian
ESP	Castilian Spanish	PTG	Portuguese
FIN	Finnish	SVE	Swedish
FRA	French		

For more information about internationalization and Windows, see the *Microsoft Windows Programmer's Reference*.

- Create an environment variable for the language, either with **Setnet32**, or, for applications that are deployed on Windows 3.1 systems, in the **informix.ini** file (by adding the variable and its setting to the [ENVIRONMENT] section). Then use the built-in **FGL_GETENV()** function of 4GL in your code. For example:

```
VARIABLE langStr CHAR(30)
LET langStr = FGL_GETENV("LANGUAGE")
```

Your program can now test for the value that you specified in **LANGUAGE**.

You can develop your own language variable scheme. For example, the following three-letter codes identify a unique subdirectory that contains the translation files appropriate for a particular language.

Subdirectory	Language
eng	English
fre	French
ita	Italian
spa	Spanish

You might want to use this or the next approach if you need to control your application's language setting separately from that of other Windows applications.

- Create your own **.ini** file and language variable, and have your application read this file for the language setting.

UNIX Environment Variables

The value of the X/Open-defined **LANG** environment variable specifies the language environment. No standardization of **LANG** locale values exists between systems. Exact values to specify for locale variables are specific to the system and also depend on which language supplements have been installed on the system.

To query programmatically for the language value, you can use the built-in `FGL_GETENV()` function:

```
FGL_GETENV ( "LANG" )
```

For more information about the **LANG** environment variable, see the *Informix Guide to SQL: Reference*.

Storing Localization Information

This section describes the process involved in creating an application so that it can read translation information, either from a file or from a database table at runtime.

File-Based Localization

You can store the translations of localized information in disk files and access them at runtime as needed.

You can use subdirectories to store language-sensitive files so they can easily be switched to create a new runtime environment. In the following example, the filename is composed by reading the value of an environment variable (created by the programmer) that specifies a Windows language subdirectory:

```
LET file001 = FGL_GETENV("LANGUAGE"), "\", "trans.4gl"
# Evaluates to "eng\trans.4gl" if LANGUAGE is "eng"
# Program reads the eng directory for copy of translation
#
# Evaluates to "ger\trans.4gl" if LANGUAGE is "ger"
# Program reads the ger directory for copy of translation
#
LET tranfile = file001
```

In the preceding example, change the backslash character to a forward slash (/) for UNIX systems.

Table-Based Localization

Localization information can also be stored in database tables. This information can be used when you initialize or run the application to change the value of variables that define titles, menus, and other language or culturally sensitive elements of the user interface. An advantage of the table-based approach is that it is highly portable between systems.

Setting Up a Table

The following example shows one way that you might set up a table to store menu options:

```
CREATE TABLE menu_elements(
  option_language  CHAR(3), #language ID code
  option_number    SMALLINT, # identifying number
  option_text      CHAR(80), # text
  option_maxlen    SMALLINT # maximum length of string
)

CREATE UNIQUE INDEX ix_menustr
  ON menu_elements(option_language, option_number)
```

Example data:

```
ENG150Cold Beer
FRE150Bière froide
GER150Kaltes Bier
SPA150Cerveza fría
ENG151Iced Tea
...
```

Querying the Table

A global variable that contains the language code of the application, which corresponds to the value in the **option_language** column, can be set in the program at startup. Each time a character string is needed, a function could be called that uses the language and identifying number to query the table for the appropriate string:

```
LET lang = getLanguage()# returns 3 letter code
# from option_language column
```

Localizing Prompts and Messages

You can use the 4GL message compiler utility to create translated message files for your application messages. These files, which usually have the extension **.iem**, run very quickly.

Creating Message Files

For any natural language, follow these steps to create new language versions of the messages and prompts that your application displays.

To create new message files

1. With a text editor that can create flat files, create a source (**.msg**) file with the following format:

```
.message-number  
message-text  
.message-number  
message-text
```

For example:

```
.1000  
Part not found.  
.1001  
Price must be a positive number.  
.1002  
Invalid format for phone number.
```

To translate the messages into another language, simply provide translated versions for the message text, using the same format.

2. At the system prompt, invoke the message compiler utility (**fglkmmessage**) by using a command of the following form:

```
fglkmmsg filename
```

The message compiler processes **filename.msg** and produces a compiled message file that has the name **filename.iem**.

If you want the compiled message file to have a different name from the source file, specify that filename as a final argument:

```
fglkmmsg source output
```

The syntax of **fglkmmessage** is described in the *INFORMIX-4GL Reference Manual*.

Accessing Message Files

To access the compiled message file from your application, you can write a function that reads the messages from the compiled (**.iem**) file. For example, the calling program includes logic to display a `Part not found` message in the following pseudo-code:

```
DEFINE OK, noPart INT, msg CHAR(79)  
LET noPart = 1000
```

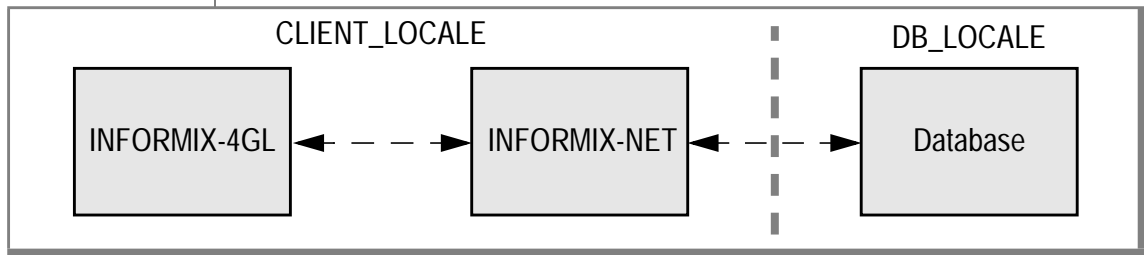
To supply new versions of the messages, you need only provide a new source file and compile it with the message compiler. The function calls in your application remain the same.

Handling Code-Set Conversion

The process of converting characters at the locale of the 4GL application to characters at the locale of the database server (or vice versa) is called *code-set conversion*. If your application needs to run on computers that encode different character sets, it might be necessary to enable code-set conversion. This section provides some background and details.

Code-set conversion is performed by INFORMIX-NET; no explicit code-set conversion is done by 4GL. [Figure D-3](#) shows the relationship between 4GL, INFORMIX-NET, and the database.

Figure D-3
Processes and Their Locales



The code sets in the **CLIENT_LOCALE** can differ from those in **DB_LOCALE**. In the **CLIENT_LOCALE**, the code sets (which are specified in locales) use code points that are pre-defined by Microsoft standards. The code sets that are used in the **DB_LOCALE** tend to use characters that are based on UNIX conventions, if the application is designed to access legacy data.

Code-set conversion is done by way of a code-set conversion file. Files for code-set conversion between **CLIENT_LOCALE** and **DB_LOCALE** need to be present on the client. For conversion to take place, conversion files need to be present in the `%informixdir%\gls\cv` directory.

For details of converting between client and server code sets, see the sections that follow. For more information, see the *Informix Guide to GLS Functionality*.

What Is Code-Set Conversion?

Different operating systems sometimes encode the same characters in different ways. For example, the character *a-circumflex* is encoded:

- in Windows code page 1252 as hexadecimal 0xE2.
- in IBM CCSID 437 as hexadecimal 0x83.

If the encoding for *a-circumflex* on the Windows system is sent unchanged to the IBM system, it will be printed as the Greek character gamma. This happens because, on the IBM system, gamma is encoded as 0xE2.

This means character data strings that are passed between two computers using different character set encodings must be converted between the two different encodings. Otherwise, character data originating from one computer will not be correctly displayed or processed on the other computer.

This appendix uses the term *code set* in the same way the Windows documentation uses the terms *character set* and *code page*.

Converting character data from one encoding schema to another is called *code-set conversion*. If a code-set conversion is required from computer A to computer B, it is also required from computer B to computer A. You must explicitly enable code-set conversion; no conversion is done by default. (Details on enabling code-set conversion appear in [“Enabling Code-Set Conversion for Windows” on page D-53.](#))

What Code-Set Conversion Is Not

Code-set conversion is not a semantic translation; that is, it does not convert words between different languages. For example, it does not convert between English *yes* and French *oui*. It only ensures that each character is processed and printed the same, regardless of how the characters are encoded.

Code-set conversion does not create a character in the target code set if the character exists only in the source code set. For example, if the character *a-circumflex* is being passed to a computer whose code set does not contain an *a-circumflex* character, the target computer will never be able to exactly process or print the *a-circumflex* character. This situation is described in more detail in [“Mismatch Processing” on page D-52.](#)

When You Do Not Need Code-Set Conversion

You do not need code-set conversion in any of the following situations:

- The client and the server are on the same computer.
- The code set of your client and of all the databases to which you are connecting are the same.
- The subset of characters that you will be sending between the client and the server are encoded identically. For example, if you are sending only English characters between a client and a server, and each English character has the same encoding on both computers, no code-set conversion is required. In this case, the non-English characters can have different encodings.
- The character-string data values are passed from the client to the server for storage only and are neither processed nor printed by the server. For example, no code-set conversion is required if a client:
 - passes character-string data to the server.
 - does not process or print the data on the server computer.
 - retrieves the same data for processing or printing on computers that use the same code set as the client that populated the database.

Sorting data by using the `ORDER BY` statement or retrieving data by using a `LIKE` or `MATCHES` clause, however, will probably produce erroneous results if the data strings are not converted before they are stored.

What Data Values Are Converted

If you enable code-set conversion, data values are converted by INFORMIX-NET PC from the 4GL client to the database server, and from the server to the client. The `CHAR`, `VARCHAR`, and `TEXT` blob data types are converted, as are column names, table names, database names, and SQL command text.

Mismatch Processing

If both code sets encode exactly the same characters, then mismatch handling is unnecessary. If the source code set contains any characters that are not contained in the target code set, however, the conversion must define how the mismatched characters are to be mapped to the target code set.

Four ways code-set conversions handle mismatch processing are as follows:

- **Round-trip conversion.** This maps each mismatched character in the source code set to a unique character in the target code set. On the return, the original character is mapped back to itself. This guarantees that a two-way conversion will result in no loss of information; however, data converted in only one direction might confuse the processing or printing on the target computer.
- **Substitution conversion.** This maps all mismatched characters in the source code set to a single specific character in the target code set that serves to highlight mismatched characters. This guarantees that a one-way conversion will clearly show the mismatched characters; however, a two-way conversion will result in information loss if mismatched characters are transferred.
- **Graphical replacement conversion.** This maps each mismatched character in the source code set to a character in the target code set that resembles the source character (this includes mapping one-character ligatures to their two-character equivalents). This might confuse printing on the target computer. Round-trip conversions should contain as many graphical replacement conversions as possible.
- **Substitution plus graphical replacement.** This maps as many mismatched characters as possible to their graphical replacements, and maps the remaining mismatched characters to the substitution character.

Informix-supplied code-set conversion source files have header comments that indicate which method was used.

The following information is specific to Windows. Information for UNIX appears in [“Enabling Code-Set Conversion for UNIX” on page D-56](#).

Enabling Code-Set Conversion for Windows

Code-set conversion on Windows is handled by INFORMIX-NET *for Windows*. There is no portable way to determine which code set an operating system is using, so you must tell INFORMIX-NET which code set is being used by all the databases to which your client will be connecting in a single connection.

For INFORMIX-NET to work correctly, all keyboard input, terminal output, and file input and output must use the same code set on the client computer. All databases to which your application connects during a single connection must also use the same code set.

Follow these steps to establish code-set conversion. Each step is described in more detail in the paragraphs that follow.

To establish code-set conversion

1. Determine the code set that is used by the client.
2. Determine the code set that is used by all the databases to which this client will connect in a single connection.
3. Determine whether you have an Informix-defined code-set conversion that is suitable for use between the client and database code sets.
4. Determine the Informix-defined code-set names that are used to identify the client and server code sets.
5. Assign the Informix-defined code-set names to the **CLIENT_LOCALE** and **DB_LOCALE** entries in the Windows 95 registry through the **Setnet32** utility, or in the **InetLogin** structure (**login.h** file). Programs deployed on Windows 3.1 can set these entries in the [Environment] section of the **informix.ini** file.
6. Launch the 4GL application.

You must modify applications that write blobs to a database to set **loc_loctype** (in the locator structure **loc_t**) to **SQLBYTE** or **SQLTEXT**. Setting this enables INFORMIX-NET to determine if you are writing a binary blob (**SQLBYTE**) that should not be converted, or a text blob (**SQLTEXT**) that should be converted. You do not need to set this parameter for reading blob data.

Determining the Code Sets Used by the Client and Database

Because each operating system has its own way of declaring the code set it is using, see your Windows system documentation or your system administrator to determine the code set that is used by the client computer.

Your system administrator should also know which code set is being used by the database.

Determining the Available Code-Set Conversions

All the code-set conversions available to you are located in the `%informixdir%\gls\cv` directory. If you have INFORMIX-NET, `%informixdir%` indicates the directory in which INFORMIX-NET is installed.

The object file for each conversion has the suffix `.cvo`. The corresponding source file for each conversion has the suffix `.cv`. You need two object files for each conversion, one for the client-to-server direction and one for the server-to-client direction. The following table lists a few examples of code-set conversion files that are currently available.

Code Sets	Conversion Files
1250 to and from 852	04E20354.cvo and 035404E2.cvo
1250 to and from ISO8859-2	04E20390.cvo and 039004E2.cvo
1251 to and from 856	04E30362.cvo and 036204E2.cvo
1251 to and from ISO8859-5	04E3E004.cvo and E00404E3.cvo
1252 to and from 437	04E401B5.cvo and 01B504E4.cvo
1252 to and from ISO8859-1	04E40333.cvo and 033304E4.cvo
1252 to and from 850	04E40352.cvo and 035204E4.cvo

Determining the Informix-Defined Name of a Code Set

Each code-set conversion source file indicates the Informix-defined names of the code sets that it converts in its header comment. Use these names in the **InetLogin** structure or (for Windows applications only) in the **informix.ini** file to tell INFORMIX-NET what conversion to perform.

The names are defined in the Informix code-set name registry file. This file is named **registry** and is located in the directory `%informixdir%\gls\cm`.

Specifying the Conversion Filenames Using INFORMIX-NET

To enable code-set conversion for INFORMIX-NET, assign the Informix-defined code-set names to the **CLIENT_LOCALE** and **DB_LOCALE** entries in the **InetLogin** structure (see the **login.h** file). For applications deployed on Windows 3.1, you can add an entry to the **informix.ini** file with the format:

```
CLIENT_LOCALE=code-set name of client machine
DB_LOCALE=code-set name of all databases
```

For example:

```
CLIENT_LOCALE=1252
DB_LOCALE=ISO8859-1
```

If your application must run in more than one locale with different code sets, it is better to set the entries programmatically in the **InetLogin** structure, rather than setting the entries with the **Setnet32** utility (for Windows 95 and Windows NT applications) or in the **informix.ini** file (for Windows 3.1 applications).

You can also set these and other database environment variables through the **Setnet32** program: start **Setnet32**, and click **MORE** to display the second page of options.

To change to a different code-set conversion, close the connection by exiting from the 4GL application. Then set new values for **CLIENT_LOCALE** and **DB_LOCALE** and restart the application.

To disable code-set conversion through the **InetLogin** structure, set **CLIENT_LOCALE** and **DB_LOCALE** to **NULL** or to the same code set. (To disable code-set conversion on Windows applications, delete the **CLIENT_LOCALE** and **DB_LOCALE** entries from the **informix.ini** file.)

To establish code-set conversion

1. Set the `SQL_TRANSLATE_DLL` parameter to the name of the DLL that contains the character translation functions.
2. Set the `SQL_TRANSLATE_OPTION` parameter to a number that indicates the current translation option.
Options are specific to the driver-specified translation DLL.

Enabling Code-Set Conversion for UNIX

Code-set conversion on UNIX is handled by UNIX environment variables.

To establish code-set conversion on UNIX

1. Determine the code set used by the client.
2. Determine the code set used by all the databases to which this client will be connecting in a single connection.
3. Specify the conversion filenames.
4. Start the application.

Determining the Code Sets Used by the Client and Database

Because each operating system has its own way of declaring the code set it is using, consult your UNIX operating system documentation or your system administrator to determine the code set used by the client computer.

Your system administrator should also know which code set is being used by the database.

Specifying the Conversion Filenames

Set the `DBAPICODE` environment variable to specify a code set that has a mapping file in the message directory `$INFORMIXDIR/msg` (or a directory pointed to by the `LANG` or `DBLANG` value). The Informix `crtcmmap` utility helps you to create mapping files.

For more information, see the *Informix Guide to SQL: Reference*.

Index

Numerics

- 4GL functions, using 6-8
- 4GL Server 12-3
- 7.x libraries, finding B-8

A

- About Box for Java applets 11-53
- a-circumflex character,
 - coding D-49
- AFTER FIELD clause, with HTML Client 10-8
- AIX operating system 2-4
- Aliases, with Java Client 11-9
- Alphanumeric characters D-24
- ALS (Asian language support) D-3, D-36
- Applet viewer, and Java Client 11-12
- Applet, definition of 11-4
- Application Server, definition 10-6
- Applications
 - terminating 5-29
 - Windows, starting 5-20
- ar42o script 4-27
- Architecture, general Dynamic
 - 4GL 1-5
- Arcs, drawing 7-25
- Arrays
 - configuration for HTML Client 10-65
 - differences from 4GL 1-7
 - displaying a row 5-24
 - with HTML Client 10-26

Asian Language Support

- (ALS) D-3, D-37
- Asian languages D-4, D-15, D-27
- Ataman 12-29
 - Installing and Configuring 2-22

B

- BEFORE FIELD clause, with HTML Client 10-8
- Bidirectional text processing D-17
- Big-5 code page D-20
- Binary blob D-52
- Bitmaps, implementing 6-11
- Blobs, text D-24
- Border width configuration 9-13
- Browser buttons, with HTML Client 10-9
- Button object 9-7
- Buttons
 - adding to a form 6-9
 - Check 9-7
 - horizontal title menu 9-7
 - implementing hot-key 6-6
 - implementing menu 6-6
 - key 9-7
 - Key BMP 9-7
 - menu 9-7
 - No Key B-15
 - relief configuration 9-13
- Byte-based string operations D-28
- bytecode 11-4

C

C code

- compiling to 4-19
- example 4-19
- libraries 2-12, 2-22

C compiler 2-6

C compiler, GNU A-8

C functions

- returning key codes from 5-18
- using in 4GL applications 4-8

C language A-7

Canadian French language

- code D-43

Canvas 9-8

CapsLock and scrollbar,

- troubleshooting B-17

Castilian Spanish language

- code D-43

CC environment variable A-7

Certificate authority, and HTML

- Client 10-47

Channel

- closing 5-8
- error codes 5-8
- extensions 5-3
- opening a file 5-4
- opening a pipe 5-5
- reading data 5-6
- setting default separator 5-6
- writing data 5-7

CHAR data type D-16

Character filter, creating 5-18

Character set D-12, D-49

Character string printable

- characters D-23

Characters, special B-10

Check boxes, implementing 6-11

Check button 9-7

Chinese language D-4, D-15, D-20

Circles, drawing 7-24

CJAC, Cli Java Application

- Connector, introduction to 11-5

CJA, Cli Java Applet 11-6

CLASSPATH environment

- variable 11-4

Cli Java Applet (CJA) 11-6

Cli Java Application Connector

- (CJAC), introduction to 11-5

Client locale D-21

Client/server architecture 1-5

CLIENT_LOCALE environment

- variable D-23, D-35, D-39, D-54

cli-html.exe file 10-14

cli-html.iem file 10-71

Closing a channel 5-8

Closing a window 7-10

Code page 1252 D-39

Code points D-12, D-48

Code set D-13

Code-set conversion

- files D-53
- handling D-48
- tables D-24

Collation order D-14, D-17, D-26

COLLCHAR environment

- variable D-17

Color settings for Java Client 11-50

Colors

- changing line 7-22
- configuration 9-10
- setting fill 7-23
- specifying drawing 7-21

Column name D-25

Combo boxes

- implementing as form
 - extension 6-14
- implementing in HTML
 - Client 10-35

COMMAND KEY options, with

- HTML Client 10-8

Comment icons Intro-7

Comments D-23

Common Gateway Interface

- (CGI) 11-5

Compatibility of servers D-3

Compilation

- sample program 3-4
- to C code 4-19
- tools for 4-26

Compiling libraries 2-12

Composite characters D-4, D-21,

- D-27

confdesi Configuration Manager

- program 9-4

Configuration B-18

- border width 9-13
- color 9-10

general Dynamic 4GL 8-3

general GUI 8-17

HTML Client 10-48

menu style 8-19

numeric fields 9-10

radio buttons 9-10

relief 9-13

settings for HTML Client 10-23

stopping, for colors 9-11

toolbars 8-20

using the Configuration

- Manager 9-4

Web applications 10-52

Windows Client 12-31

Configuration file

HTML Client 10-70

Windows Client 12-37

Configuration Manager,

running 9-4

Constant name D-25

Constraint name D-25

Contact information Intro-8

crtcmmap utility D-55

Currency symbols D-28

Cursor name D-25

Cursor position

- returning 5-27
- setting 5-28

Cursors scope range 1-8

Customizing the Windows

- Client 12-31

Cyrillic alphabet D-20

D

Danish language code D-43

Data types

- CHAR D-16
- NCHAR D-14, D-16, D-17
- NVARCHAR D-14, D-16, D-17
- VARCHAR D-16

Datetime 1-7

DBAPICODE environment

- variable D-35

DBASCIIBC environment

- variable D-36

DBCODASET environment

- variable D-36

DBCONNECT environment variable D-36
 DBCSOVERRIDE environment variable D-36
 DBDATE environment variable D-35
 DBFORMAT environment variable D-35, D-40
 DBLANG environment variable D-40
 DBMONEY environment variable D-35, D-40
 DBNLS environment variable D-17
 DB_LOCALE environment variable D-35, D-38, D-54
 DDE
 definition 1-10
 using 5-8
 Debugger D-22
 Decimal point D-40
 Dependencies, software Intro-5
 Deployment D-5, D-22
 DG/UX operating system 2-4
 Diacritical marks D-13
 Dialog boxes, creating 7-12
 Differences from 4GL 1-7
 Digital UNIX 2-5
 Disabling password display, for HTML Client 10-19
 Disk space requirements 2-6
 DISPLAY 1-9
 DISPLAY ARRAY statement
 extension of 5-14
 new triggers for 1-11
 DISPLAY environment variable 8-9, 9-4
 Display extensions to 4GL 7-3
 DISPLAY statement
 with HTML Client 10-9
 Display width D-28
 Displaying installation options 2-8
 Documentation, types of Intro-7
 DOS naming conventions, troubleshooting B-12
 Drawing area, selecting 7-20
 Drawing extensions to 4GL 7-17
 Dutch language code D-43
 DVM, Dynamic Virtual Machine 11-3

Dynamic Virtual Machine (DVM) 11-3
 Dynix/Ptx operating system 2-5

E

East Asian languages D-27
 Eight-bit clean D-12
 Email, with HTML Client applications 10-36
 emm386, enhancing performance B-19
 Emulator, UNIX, starting 5-19
 Enhancements for HTML Client 10-9
 envfcomp file 3-3
 Environment settings D-21
 Environment shell script, creating 2-13
 Environment variables
 CC A-7
 CLASSPATH 11-4
 CLIENT_LOCALE D-23, D-35, D-39, D-54
 COLLCHAR D-17
 DBAPICODE D-35
 DBDATE D-28, D-35
 DBFORMAT D-28, D-35, D-40
 DBLANG D-40
 DBMONEY D-35, D-40
 DBNLS D-17
 DB_LOCALE D-38
 DISPLAY 8-9, 9-4
 Dynamic 4GL Product A-1
 FGLCC 3-3, A-4
 FGLDBPATH A-2
 FGLDBS 3-3
 FGLDEBUGON A-7
 FGLDIR 3-3, A-3
 FGLGUI 1-12, 3-4, 7-4, 13-10, A-2
 FGLLDPATH A-5
 FGLLIBSQL 3-4, A-5, B-9, B-10
 FGLLIBSYS 3-4, A-6, B-10
 FGLRUN A-4
 FGLSERVER 8-9, 13-6
 FGLSHELL 3-4
 FGLSQLDEBUG A-6
 for TCL/TK 13-5

GCC A-8
 GCCDIR A-8
 GCC_EXEC_PREFIX A-8
 INFORMIXC 2-6
 INFORMIXDIR 3-3
 INFORMIXHOST 8-13
 INFORMIXPROTOCOL 8-12
 INFORMIXSERVER 8-13
 INFORMIXSERVICE 8-12
 INFORMIXSQLHOSTS 8-13
 LANG D-44
 LC_COLLATE D-17, D-26
 LD_LIBRARY_PATH 3-4
 PATH 3-4, A-3, A-9, A-10
 returning the value D-43
 SERVER_LOCALE D-36
 setting 3-3
 setting for the compiler 4-3
 setting in Windows Registry D-21
 setting through Setnet32 D-43
 TCLDIR A-9
 TCL_LIBRARY A-10
 TK_LIBRARY A-10
 Windows system language variables D-42
 WINSTATIONNAME 8-9
 envtcl shell script 13-8
 en_us.1252@dict D-15, D-17
 en_us.1252@dict locale D-26
 en_us.8859-1 D-17
 Error messages and internationalization D-34
 ERROR statement, with HTML Client 10-9
 ESQL/C 2-6
 Example C-code program 4-19
 Example cjac.cnf file 11-45
 Example P-code program 4-6
 Extended ASCII character sets D-12
 Extensions to the 4GL language 6-3

F

fgl2c 4-27, 8-5, A-3, B-17
 fgl2cres.web file 10-71
 fgl2p 4-27, A-4

FGLCC environment variable 3-3, A-4
 fglcl file 10-16
 fglcl.conf file 10-70
 fglcomp program 4-27
 FGLDBPATH environment variable A-2
 FGLDBS environment variable 3-3
 FGLDEBUGON environment variable A-7
 FGLDIR environment variable 3-3, A-3
 fglfontsel program 4-28
 fglfontsel.42e program 4-28
 fglform compiler 1-12, 4-27
 FGLGUI environment variable 1-12, 3-4, 7-4, 13-10, A-2
 fglhtml file 10-16
 fglhtml HTML server process 10-71
 fglinstall script 4-28
 FGLLDPATH environment variable A-5
 FGLLIBSQL environment variable 3-4, A-5, B-9, B-10
 FGLLIBSYS environment variable 3-4, A-6, B-10
 fgl link program 4-27
 fglmkmsg program 4-28
 fglmkmsg utility D-4, D-7
 fglmkrun script 4-11
 fglnodb runner 4-27
 fglpager 1-11
 fglpager command 1-11
 fglpager script 4-28
 fglprofile file editing for buttons 6-7
 fglprofile.web file 10-71
 FGLRUN environment variable A-4
 fglrun runner 4-27, 10-6
 fglschema A-2
 fglschema script 4-27
 FGLSERVER environment variable 8-9, 13-6
 FGLSHELL environment variable 3-4

FGLSQLDEBUG environment variable A-6
 fglWrt program 4-28
 fgX11d 8-9, 13-5, A-7
 fgX11d daemon 4-28
 FGL_GETENV() function D-43, D-44
 Field names, returning 5-23
 Field values returning 5-23 setting 5-23
 Fields relief configuration 9-13 retrieving information from 7-8 returning a value after modification 5-21
 File extensions .cv D-53 .cvo D-53 .iem D-47 .ini D-42, D-44 .msg D-32
 Filename D-25
 File, opening with channels 5-4
 Fill color, setting 7-23
 Filtering router, and HTML Client 10-46
 Filter, creating a custom character 5-18
 findlib.sh script 4-27
 Finnish language code D-43
 Firewall, and HTML Client 10-46
 Font requirements D-14
 Fonts for Java Client 11-52
 Formal argument D-25
 Forms, compiling 4-4 4gluser.msg file D-32
 French language D-33
 French language code D-43
 Function name D-25
 Functions, using 4GL 6-8

G

GB 2312-80 code page D-20
 GCC A-8
 GCC compiler, and SCO B-14
 GCC environment variable A-8

GCCDIR environment variable A-8
 GCC_EXEC_PREFIX environment variable A-8
 German language code D-43
 Glossary of localization terms, keeping D-33
 GLS installation 2-11 servers D-3
 GL_DATE environment variable D-35
 GL_DATETIME environment variable D-35
 GNU C compiler 2-6, A-8
 Graphical replacement conversion D-51
 Greek characters D-49
 Greek language D-20
 GUI configuration settings 8-17

H

Hardware requirements 2-5
 Headers and footers, HTML Client application 10-19
 Help messages compiling 4-4 translating D-27
 High-order bit D-12
 Horizontal title menu button 9-7
 HPUX operating system 2-4
 HTML Client architecture 10-6 configuration 10-48 definition of 10-16 enhancements 10-9 enhancing application interface 10-18 example 10-72 implementing combo boxes 10-35 installing 10-9 limitations 10-8 manual UNIX installation 10-69 sample application 10-20 sample configuration settings 10-23 sample forms 10-37 security levels 10-44

server message file 10-71
 troubleshooting NT
 installation 10-73
 HTML Emulation for tables 10-41
 HTML server 10-16
 HTML tags 10-19
 httpd file 10-16
 httpd server 10-6

I

Icelandic language code D-43
 Icon modification D-27
 Icons
 Important Intro-7
 Tip Intro-7
 Warning Intro-7
 Identifiers D-24
 Important paragraphs, icon
 for Intro-7
 Index name D-25
 InetLogin structure D-52, D-54
 INFORMIXC environment
 variable 2-6
 INFORMIXDIR environment
 variable 3-3
 INFORMIX-ESQL/C D-13
 INFORMIXHOST environment
 variable 8-13
 INFORMIX-NET D-48, D-54
 INFORMIX-NET PC D-3
 INFORMIXPROTOCOL
 environment variable 8-12
 INFORMIX-SE database server D-3
 INFORMIXSERVER environment
 variable 8-13
 INFORMIXSERVICE environment
 variable 8-12
 INFORMIXSQLHOSTS
 environment variable 8-13
 informix.ini file D-54
 Installation
 Dynamic 4GL 2-3
 hardware requirements 2-5
 HTML Client 10-9
 Java Client 11-14
 reinstalling Windows Client B-12
 TCL/TK 13-4

Windows Client 12-5
 X11 Client 13-4
 install.sh script 4-28
 Internal data file corrupted error,
 troubleshooting B-13
 International Language
 Supplement D-19
 Internationalization
 codeset conversion D-48
 enabling for UNIX D-55
 enabling for Windows D-52
 definition D-12
 fonts D-31
 keyboard layouts D-31
 measurement systems D-30
 messages D-46
 overview of methodologies D-34
 paper size D-31
 reports D-32
 translation checklist D-32
 IRIX operating system 2-5
 ISO 8859 D-12
 ISO Standard A4 D-31
 Italian language code D-43

J

JA 7.20 supplement D-19
 Japanese language D-4, D-15, D-19,
 D-20
 Java Client
 architecture 11-3
 enhancements 11-57
 installing 11-14
 limitations 11-8
 Java Foundation Classes (JFC) 11-5
 Java, introduction to 11-3
 Joins D-37

K

KEY 1-8, 8-28
 Key binding, removing 7-19
 Key BMP button 9-7
 Key button 9-7
 Key code values, returning 5-15
 Key code, invoking 6-13
 KEY Field attribute, setting 6-8

Kinsoku processing D-27
 KO 7.20 supplement D-19
 Korean language D-4, D-19, D-20
 KSC 5601 code page D-20

L

LANG environment variable D-44
 Language codes D-43
 Language-sensitive files D-45
 Language supplement D-19
 Language variable D-42
 Latin alphabet D-20
 LC_COLLATE environment
 variable D-17, D-26
 LD_LIBRARY_PATH environment
 variable 3-4
 Length of identifiers D-24
 Libraries, 7.x, finding B-8
 Libraries, system, finding B-10
 licencef4gl script 4-28
 License configuration 8-14
 Line width, setting 7-22
 Line, drawing 7-24
 Link errors on Windows B-12
 Linking modules 4-7, 4-21
 Links between pages, with HTML
 Client 10-40
 Links, with HTML Client
 applications 10-36
 Linux operating system 2-5
 List boxes 6-4
 Locale ID D-42
 Locale variables D-44
 Locales
 client D-21, D-35, D-39, D-54
 server D-21, D-35, D-38, D-54
 Localization
 defined D-12
 guidelines D-32
 Localized collation order D-14
 Logfile names D-24, D-25
 Logical characters D-14, D-28
 Logical-character-based
 operations D-28
 login.h file D-52
 Lossy conversion D-49

M

Macros, and HTML Client 10-27
 Manual Unix Installation, for
 HTML Client 10-69
 Mapping files D-55
 Memory fault,
 troubleshooting B-18
 Menu 9-9
 Menu button 9-7
 Menus
 configuring window area 9-9
 default keys 8-34
 differences from 4GL 1-8
 in multibyte locales D-27
 style configuration 8-19
 Message box, creating 7-12
 Message Compiler D-4, D-7, D-22
 Message file, HTML server 10-71
 MESSAGE statement, with HTML
 Client 10-9
 Microsoft Windows Programmer's
 Reference D-32
 Mismatch handling D-51
 Modern Spanish language
 code D-43
 Modifying fields and returning a
 value 5-21
 Mouse management functions 7-18
 Mouse usage 1-8
 Mouse, and Numlock on X11 B-16
 Multibyte locale D-28

N

Named values D-24
 Native Language Support
 (NLS) D-16, D-32, D-37
 NCHAR data type D-14, D-16,
 D-17
 New features in Dynamic 4GL 1-12
 NEXT FIELD clause, with HTML
 Client 10-8
 NLS functionality D-32
 NLS servers D-16
 No Key buttons B-15
 Non-ASCII characters D-23

Non-composite Thai
 characters D-4, D-21
 Non-English characters D-50
 Nonprintable characters D-23
 Norwegian language code D-43
 ns-httpd file 10-16
 Numeric fields
 configuration 9-10
 Numlock, troubleshooting B-16
 NVARCHAR data type D-14, D-16,
 D-17

O

ON KEY actions, with HTML
 Client 10-8
 On-line manuals Intro-7
 Opening a file with channels 5-4
 Opening a pipe with channels 5-5
 Operating systems supported 2-4
 OSF operating system 2-5
 Ovals, drawing 7-23

P

P code
 compiling to 4-6
 creating libraries 2-13
 example 4-6
 returning key codes from 5-16
 pager.42e command 1-11
 pager.42e script 4-28
 Paper size D-31
 Partial characters D-28, D-29
 Partitioned application D-21
 PATH environment variable 2-6,
 3-4, A-3, A-9, A-10
 Pathname D-25
 Paths, and Java Client 11-9
 People's Republic of China D-19,
 D-20
 Pipe
 opening 5-5
 writing data to 5-7
 Polygons, drawing 7-26
 Portuguese language code D-43
 Prepared statement name D-25
 Printable characters D-23

Printed manuals Intro-7
 Programs, Java 11-4
 Progress Bar for Java applets 11-53
 PROMPT statement
 differences from 4GL 1-9
 with HTML Client 10-8

Q

Quoted string D-24

R

Radio buttons
 configuration 9-10
 implementing 6-11
 Reading data from opened
 channel 5-6
 Rectangles, drawing 7-23
 Registry file D-54
 Registry, setting environment
 variables in D-21
 Relational operators D-14, D-26
 Reliant UNIX 2-5
 Relief configuration 9-13
 Remove key binding 7-19
 Report name D-25
 Report pager 1-11
 Reports, differences from 4GL 1-8
 Retrieving information from a
 field 7-8
 Retrieving information from a
 window 7-8
 Returning a field value 5-23
 Returning cursor position 5-27
 Returning field names 5-23
 Returning key code values 5-15
 Returning value of an environment
 variable D-43
 Returning values after
 changes 5-21
 Returning values after mouse
 click 7-18
 Rlogin, troubleshooting B-11
 Round-trip conversion D-51
 rtsinstall script 4-28

RUN WITHOUT WAITING
 statement, with HTML
 Client 10-9
 Runner environment, for HTML
 Client 10-65
 Runner, building 4-10
 Russian language D-33

S

Schema file, generating 4-5
 SCO Open Server 5, and GCC
 compiler B-14
 SCO server, troubleshooting B-12
 SCO UNIX operating system 2-4
 Screen array, displaying a row 5-24
 Screen record without size 1-12
 Screens, relief configuration 9-13
 Scrollbar and capslock,
 troubleshooting B-17
 Scrolling fields, implementing 6-15
 Security levels, and HTML
 Client 10-44
 Security, troubleshooting for
 HTML applications 10-47
 Separator, setting default with
 channels 5-6
 Server compatibility D-3
 Server locale D-21
 SERVER_LOCALE environment
 variable D-36
 servlets 11-5
 Servlets, and Java client 11-6
 Setnet32 utility D-21, D-43, D-54
 Setting compiler environment
 variables 4-3
 Setting cursor position 5-28
 Setting field values 5-23
 Shift-JIS code page D-20
 Single-byte locale D-28
 SINIX operating system 2-5
 Size, setting default for window 7-6
 SLEEP statement with HTML
 Client 10-8
 Slow rlogin, troubleshooting B-11
 Software dependencies Intro-5
 Solaris operating system 2-5
 Sorting data

in a query D-26
 in a report D-26
 Spanish language D-33
 Spawning methods
 for HTML Client 10-26, 10-63
 Special characters B-10
 SQL identifiers D-24
 SQLBYTE data type D-52
 sqlxlit statement 1-9
 SQLTEXT data type D-52
 SQL_TRANSLATE_DLL
 parameter D-55
 SQL_TRANSLATE_OPTION
 parameter D-55
 SSL, and HTML Client 10-46
 Starting a Windows program from
 UNIX B-19
 Statement label D-25
 Statically linked runner,
 building 4-10
 Stored procedure D-25
 Streams
 troubleshooting B-14
 writing data to 5-7
 Strings
 character D-23
 quoted D-24
 Substitution conversion D-51
 Substrings D-29
 Supported operating systems 2-4
 Swedish language code D-43
 Swing, and Java 11-5
 Synonym, SQL identifier D-25
 System libraries, finding B-10
 System requirements
 database Intro-5
 software Intro-5

T

Table name D-25
 Table-based localization D-45
 Table, with HTML Client 10-39
 Tag Words, and Java Client 11-9
 Taiwanese D-19, D-20
 TCLDIR environment variable A-9
 TCL/TK interpreter A-9
 Tcl/Tk interpreter 12-3, 13-3, A-9

TCL_LIBRARY environment
 variable 13-5, A-10
 TCP/IP 8-17, 12-3, 13-3, B-7, B-12
 requirements for installation 2-5
 Terminating applications 5-29
 Text blobs D-24, D-52
 Text geometry D-27
 Text insertion point,
 specifying 7-21
 Text labels D-27
 Text, drawing 7-25
 TH 7.20 supplement D-19
 Thai language D-4, D-21, D-27
 th_th.thai620 D-4, D-21
 Tip icons Intro-7
 Title, setting for window 7-7
 TK_LIBRARY environment
 variable 13-5, A-10
 Toolbars
 4GL extension 1-11
 configuration settings 8-20
 creating 7-11
 Translation D-34
 as part of localization D-12, D-32
 checklist D-33
 Troubleshooting NT, for HTML
 Client 10-73
 Turkish language D-20

U

Underscore (_) symbol D-24
 UNIX system language
 variables D-44
 UNIX-based servers D-3, D-48
 Unixware operating system 2-4
 Upgrading Dynamic 4GL 2-4
 Uppercase letters D-26
 U.K. English language code D-43
 U.S. English language code D-13,
 D-43

V

VARCHAR data type D-16
 Variable name D-25
 View name D-25

W

Warning icons Intro-7
 Web browser, and Java Client 11-11
 Web Deployment 10-6
 installation on UNIX 10-9
 Web server 10-6, 10-16
 Western European languages D-20
 White-space characters D-14, D-24, D-29
 Window management functions 7-6
 Windows applications, starting 5-20
 Windows Client
 architecture 12-3
 checking for 7-5
 command line features 12-22
 configuration 12-37
 customizing 12-31
 DOS naming conventions B-12
 example 12-15
 link errors B-12
 reinstalling B-12
 SCO server B-12
 security features 12-20
 troubleshooting B-11
 Windows help facility D-4, D-22, D-27
 Windows program, starting from UNIX server B-19
 Windows, troubleshooting on B-17
 Window, closing 7-10
 Window, retrieving information from 7-8
 Window, setting active 7-10
 Window, setting default size 7-6
 Window, setting title 7-7
 WINSTATIONNAMEenvironment variable 8-9
 Word length D-30
 Workarounds for common problems B-1
 WTK Client 12-3, 12-5
 WTK interpreter 10-7

X

X11 8-9, 8-34, 8-35, A-7
 X11 Client, example 13-8
 X11 daemon 13-3
 X11 problems, troubleshooting B-16
 X11, numlock and the mouse B-16
 X/Open D-20

Z

ZHCN 7.20 supplement D-19
 ZHTW 7.20 supplement D-19

Symbols

.bmp fields, and combo fields 6-14
 .class files 11-3
 .jar files 11-3
 .per file
 compared to .html file 10-19
 editing for buttons 6-7
 .zip files 11-3