Forms Server Release 6*i*

Deploying Forms Applications to the Web with Oracle Internet Application Server

for Windows and UNIX

May 2000

Part No. A83591-01

This book contains the information you need to deploy Forms applications to the Web using the Oracle Internet Application Server.

ORACLE

Deploying Forms Applications to the Web with Oracle Internet Application Server, for Windows and UNIX

Part No. A83591-01

Primary Authors: Tony Wolfram, Cathy Godwin

Contributing Author: Joan Carter

Contributors: Ken Chu, Steve Button, Chris Barrow, Nigel Ferris, Alex Bryant, Hubert Bakker, Duncan Mills

Contents

8 Migrating Legacy Applications to the Web

9 Network Considerations

12 Load Balancing Considerations

13 Oracle Enterprise Manager Forms Support

A Forms Server Parameters

B Oracle JInitiator

Send Us Your Comments

Deploying Forms Applications to the Web with Oracle Internet Application Server for Windows and UNIX

Part No. A83591-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available).

You can email your comments to us by sending them to oddoc@us.oracle.com.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

Deploying Forms Applications to the Web with Oracle Internet Application Server

Intended Audience

This manual is intended for software developers who are interested in deploying Forms applications to the Web with the Oracle Internet Application Server.

Structure

This manual contains the following chapters and appendices:

Chapter 1 **Introduction**

Explains the benefits of deploying applications to the Web.

Chapter 2 **Overview of Forms Server**

Introduces you to the deployment tools that you will be using by providing an overview of Forms Server architecture and its components.

Chapter 3 **Preview of Configuration Choices**

Presents a preview of configurations choices that you will face when deploying applications to the Web.

Chapter 4 **Installing Forms Server**

Describes Forms Server's installation through the Oracle Universal Installer.

Chapter 5 **Configuring the Forms Server**

Describes the steps necessary to manually configure your network environment to support the Forms Server.

Chapter 6 **Deploying Forms to the Web**

Describes the steps you must perform to deploy your applications to the Web, such as creating the executable files and broadcasting the application's URL.

Chapter 7 **Application Design Considerations**

Contains guidelines and tips for designing Forms applications for Web deployment and includes some feature restrictions.

Chapter 8 **Migrating Legacy Applications to the Web**

Includes guidelines to migrate your current applications from client/server-based or OAS cartridge implementation to Web-based Forms Server implementation.

Chapter 9 **Network Considerations**

Describes the networking implementations upon which you can deploy Web applications, and the things you need to consider when deploying Web applications on each type.

Chapter 10 **Security Considerations**

Describes common security issues that you must consider when setting up Forms Server in a networked environment.

Chapter 11 **Performance Tuning Considerations**

Describes the tuning considerations when you deploy an application over the Internet or other network environment using the Forms Server.

Chapter 12 **Load Balancing Considerations**

Discusses load balancing techniques using CGI-based load balancing.

Chapter 13 **Oracle Enterprise Manager Forms Support**

Describes the Oracle Enterprise Manager (OEM) system management tool.

Chapter 14 **Capacity Planning Considerations**

Explores the scalability features of Forms Server.

Chapter 15 **Troubleshooting Solutions**

Contains information about troubleshooting solutions for the Forms Server.

Appendix A **Forms Server Parameters**

Describes the parameters that you use to configure Forms Server.

Appendix B **Oracle JInitiator**

Describes the benefits of using Oracle JInitiator as a plug-in for your users' Web browsers.

Appendix C **AppletViewer**

Describes the AppletViewer as an alternative to using Oracle JInitiator to view applications running on the Forms Server.

Related Documents

For more information, see the following manuals:

- *Oracle Forms Developer 6i Release Notes*
- *Oracle Forms Developer: Getting Started (Windows 95/NT)*
- *Oracle Reports Developer: Publishing Reports*
- *Oracle Forms Developer and Oracle Reports Developer: Guidelines for Building Applications*
- *Oracle Forms Developer: Form Builder Reference*

Part I

Deploying Forms Applications to the Web

1 Introduction

1.1 The Internet Changes Everything

With the development of vast virtual outlets for products and information without the traditional cost of real estate, construction, or distribution, the Internet is revolutionizing business and the underlying technology that supports it.

1.1.1 Improvements in Business

The Internet is creating new business opportunities and altering the way we perform common activities, such as shopping, getting directions, managing bank and stock market accounts, tracking down telephone numbers and old friends, and getting news and information.

Businesses have moved many of their internal transactions to their intranets, making the gathering and distribution of time-sensitive information as simple as publishing a URL. Interdependent companies, such as travel agencies and airlines, have moved from a time-consuming, labor-intensive telephone model to a much less costly and far more efficient self-service model via an extranet. Businesses are using the Web to talk to each other and to their customers more directly and at a lower cost, increasing both responsiveness and profitability.

The Internet expands market opportunities, improves business processes by introducing cost-reducing efficiencies, and offers new ways to attract and retain customers.

1.1.2 Improvements in the Underlying Technology

Aside from the monumental business advantage of distributing information and services to a world-wide audience with comparatively minute overhead costs,

many additional benefits result from Web application development and deployment. These include:

- **Deployment of new versions is easier, faster, and cheaper.** To roll out a Web application, simply give users the application's URL. This distribution method reduces the time, cost, and complexity of deploying applications to a large or geographically-dispersed user base by eliminating the need to install application software on each user's desktop machine.
- **Centralized distribution means lower total cost of ownership.** Web deployment dramatically reduces the cost of administration, maintenance, and network while increasing information accessibility. Instead of multiple outposts providing system administration support, system maintenance and administration is performed from one central location. With Web deployment, application complexity moves off of each user's desktop and onto centrally located, professionally managed application servers. This makes possible professional management of your site on a small number of servers, vastly simplifying, accelerating, and standardizing maintenance tasks and dramatically lowering costs.
- **Standards-based development means better integration.** Internet application development adheres to the same industry standards (such as Java, Enterprise JavaBeans, HTML, XML, CORBA, HTTP, and so on). Common language means easier and faster integration of newly or separately developed applications.
- **Component-based development means increased productivity, easy maintenance, and reusability.** Customize applications rapidly in response to the different requirements of a diverse audience. Business developers need only alter affected components and not the entire application. Commonly applied components can easily be reused in other applications. These are just some of the ways organizations are able to respond in "Web time" to user requirements.

1.2 The Oracle Internet Platform

A typical client/server architecture involves two tiers: the client tier, which hosts the application; and the server tier, which hosts the enterprise database against which the application is run. In contrast, the Oracle Internet platform involves three tiers:

- The database tier, where enterprise data is stored
- The server tier, which hosts the application and provides many other services, including Web support

- The client tier, which hosts the browser where the application is displayed

The Oracle Internet platform is all about distributing information to users while centralizing application complexity on a small number of professionally managed servers. It is comprised of database servers, application servers, development tools, and management tools. It is *the* integrated platform for Web development, deployment, and maintenance of e-business solutions. It's based on open Internet standard interfaces and protocols, including Java, Enterprise JavaBeans, CORBA, HTML, and XML.

Through a standard Internet browser, the Oracle Internet platform manages and processes any type of information content, including text, images, Web pages, video, sound, and e-mail.

Web-based applications built on the Oracle Internet platform scale to terabytes of data and millions of users, provide around-the-clock, year-round reliability, and incorporate the leading security standards for data encryption and integrity.

The Oracle Internet platform is the lowest-cost deployment platform because it simplifies the delivery and management of applications. For example:

- Server scalability means fewer servers for lower cost and easier management.
- Server-side application and data processing mean efficient network utilization.
- On the client side, all you need is a browser: there are no incremental software costs for desktops connecting to a database.

1.3 Oracle Internet Application Server

An important component of the Oracle Internet platform is the Oracle Internet Application Server. This application server is optimized to deploy Oracle Forms applications in a multi-tiered environment. Oracle Internet Application Server is a scalable, secure, middle-tier aplication server. It enables you to deliver web content, host web applications, and conntect back-office applications.

Oracle Internet Application Server is available in two versions: the Standard Edition and the Enterprise Edition. The Standard Edition is appropriate for smaller websites that require minimal transaction processing capablitiy. The Enterprise Edition is recommended for medium to large sized websites that handle a high volume of transactions.

The Oracle Internet Application Server Enterprise Edition contains Oracle Forms Server, which delivers the application infrastructure and the event model to ensure that Internet-based Forms applications automatically scale and perform over any

network. Built-in services provided by Oracle Internet Application Server include transaction management, record caching, record locking, exception handling, and load balancing. Business developers don't have to implement low-level code to build these shared services, Oracle Internet Application Server automatically delivers them as part of its own engine.

The application is optimized to reduce network traffic for Internet, intranet, and extranet deployments. Through Oracle Internet Application Server, organizations can choose to deploy via socket-based TCP/IP communication between the Java client and the server and via HTTP 1.1 (which supports firewalls) between the Java client and the server, depending on their system architecture and requirements.

Tight integration with the database server means smooth, reliable communication between the Oracle Internet Application Server engine and the database, in particular thanks to array processing and stored procedures.

There are many additional benefits to be realized from using Oracle Internet Application Server. Here are just a few:

- **Extensible optimized Java client.** Business developers can incorporate JavaBeans and reuse Java classes in their Forms applications. This extends the client Java applet and enables business developers to build really sophisticated user interfaces. These interfaces leverage the strengths of the Java language and allow for the reuse of existing Java components.
- **Automatic scalability over any network.** Oracle Internet Application Server natively delivers load balancing capabilities. Load balancing efficiently distributes client requests across available system resources. The application offers the same level of scalability with any web server used as an HTTP listener with the same level of scalability. It is optimized for corporate intranet, extranet, and Internet deployment. You can use the application on LAN, WAN, and dial-up network architectures.
- **Built-in optimizations for high performance.** Oracle Internet Application Server has many built-in optimizations that work around the two main constraints in typical three-tier architectures: network bandwidth and latency between the client and application server.

  The application reduces network bandwidth by intelligently condensing the data stream using advanced algorithms.

  One way Oracle Internet Application Server tackles latency is through Event Bundling: When a user navigates from item A to item B (such as when tabbing from one entry field to another), a range of pre- and post-triggers may fire, each of which requires processing on the server. Event Bundling "gathers" all the

events triggered while navigating between the two objects and delivers them to the server as a single packet for processing. When navigation involves traversing many objects (such as when a mouse click is on a distant object), Event Bundling gathers all events from all of the objects that were traversed and delivers them as a single network message to the server.

- **Integration with a highly productive, declarative Rapid Application Development (RAD) tool.** Oracle Internet Application Server was developed specifically to serve Oracle Forms applications. This simplifies the transition from development to deployment by eliminating time-consuming issues that can arise when integrating applications and servers created with tools from disparate vendors.

1.4 How This Guide Can Help

When you choose to deploy applications to the Internet, there are many decisions to be made as to how you will go about it. This guide provides information about those decisions and offers suggestions and methods for configuring your system for Web deployment of your applications.

We provide:

- An overview of Forms Server architecture.
- A guide for installing and configuring Forms Server in a variety of Web deployment scenarios
- A section on migrating your legacy client/server applications to the Web
- Sections on capacity planning and load balancing to help you set up multiple servers that work and communicate together to share growing workloads
- Sections on network and security considerations
- Sections on application design considerations and performance tuning for optimizing the performance of your Web applications

2

Overview of Forms Server

2.1 Introduction

The Oracle Internet Application Server is a scalable, secure, middle-tier aplication server. It enables you to deliver web content, host web applications, and connect to to back-office applications. Forms Server is an integral part of the Oracle Internet Application Server bundle, which provides the technology to fully realize the benefits of Internet computing. This chapter provides an overview of Forms Server architecture, specifically as it relates to deploying forms over the Internet.

Forms Server is a new generation of development tools that enable you to deploy new and existing Oracle Forms applications on the World Wide Web. You can deploy applications on an internal company intranet, an external company extranet, or on the Internet.

Forms Server is an application server optimized to deploy Oracle Forms applications in a multi-tiered environment. It takes advantage of the ease and accessibility of the Web and elevates it from a static information-publishing mechanism to an environment capable of supporting complex applications.

2.2 Forms Server Architecture

Forms Server uses a three-tier architecture to deploy database applications. Figure 2–1 shows the three tiers that make up the Forms Server architecture:

- The **client tier** contains the Web browser, where the application is displayed and used.
- The **middle tier** is the application server, where application logic and server software are stored.
- The **database tier** is the database server, where enterprise data is stored.

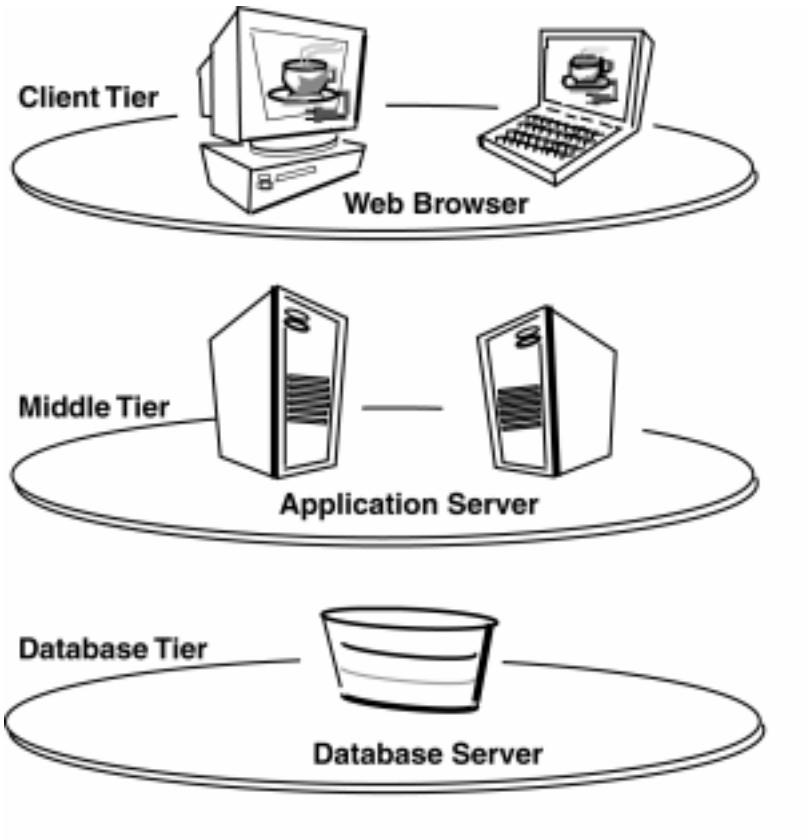


Figure 2–1 Forms Server architecture

2.3 Forms Server Components

The Forms Server is a middle-tier application server for deploying complex, transactional forms applications to the Internet. Developers can build new applications with Oracle Forms Developer and deploy them to the Internet with the Forms Server. Developers can also take existing applications that were previously deployed in client/server and move them to a three-tier architecture without changing the application code.

The Forms Server consists of three major components, as shown in Figure 2–2:

- The **Forms Applet**, which is automatically downloaded to the client and viewed within the Web browser
- The **Forms Listener**, which resides on the middle tier
- The **Forms Runtime Engine**, which also resides on the middle tier

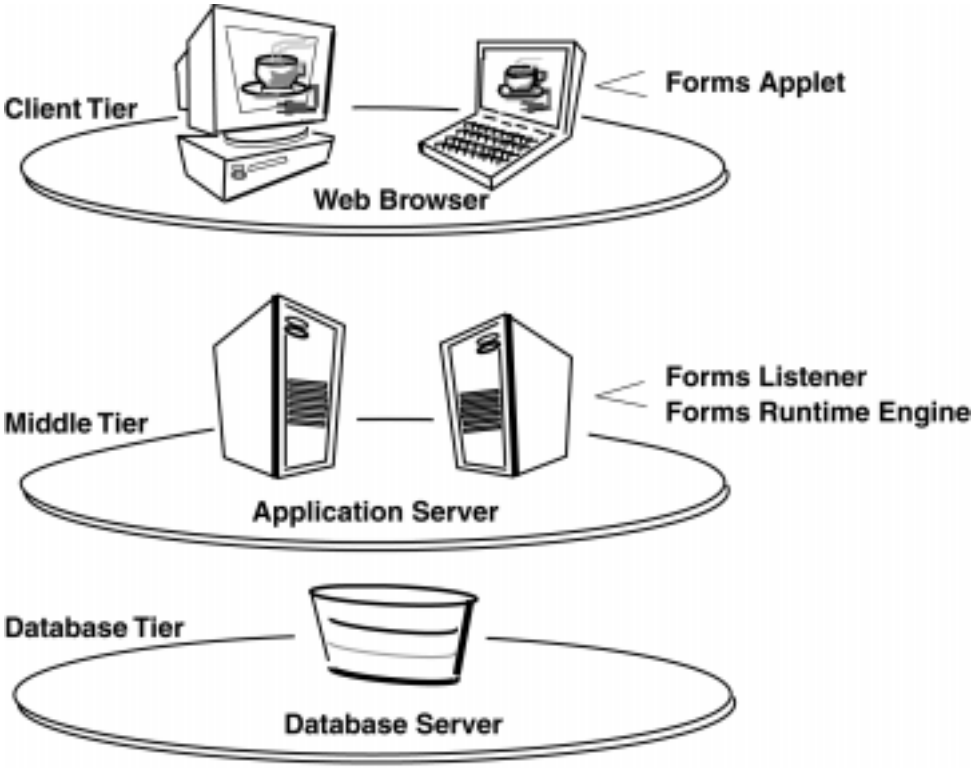


Figure 2–2 Three-tier configuration for running a form on the Web

2.3.1 Forms Applet

When a user runs a Forms session over the Web, a thin Java-based Forms applet is dynamically downloaded from the application server and automatically cached on the Java client machine.

The Forms applet provides the user interface for the Forms Server Runtime Engine. As an extensible, optimized Java applet, it operates inside the framework of the client's Web browser. It handles user interaction and visual feedback, such as information that is generated when navigating between items or when checking a check box. It is responsible for rendering the application display and contains no specific application logic.

The same Java applet code can be used for any Form, regardless of size or complexity. This means that you do not have to write Java code for every application or Form that you want to deploy on the Web.

2.3.2 Forms Listener

The Forms Listener acts as a broker between the Java client and the Forms Server runtime process. It takes connection requests from Java client processes and initiates a Forms Server Runtime process on their behalf. The listener can also maintain a pool of running engines that stand ready to make the connection from the Java client complete as quickly as possible.

2.3.3 Forms Runtime Engine

The Forms Runtime Engine manages application logic and processing. It maintains a connection to the database on behalf of the Java client. It uses the same Forms, Menus, and Libraries files that are used for running in client/server mode. No application code changes are required to deploy a legacy client/server application to the Internet.

The Forms Runtime Engine plays two roles: when it is communicating with the client browser, it acts as a server by managing requests from client browsers; when it is communicating with the database server, it acts as a client by querying the database server for requested data.

2.4 Forms Server in Action

To start and run a Forms application on the Web, users will employ a Java-enabled Web browser to access a URL. Figure 2–3 and the text that follows show and explain the sequences of events that occur during the process flow involving the Forms Server.

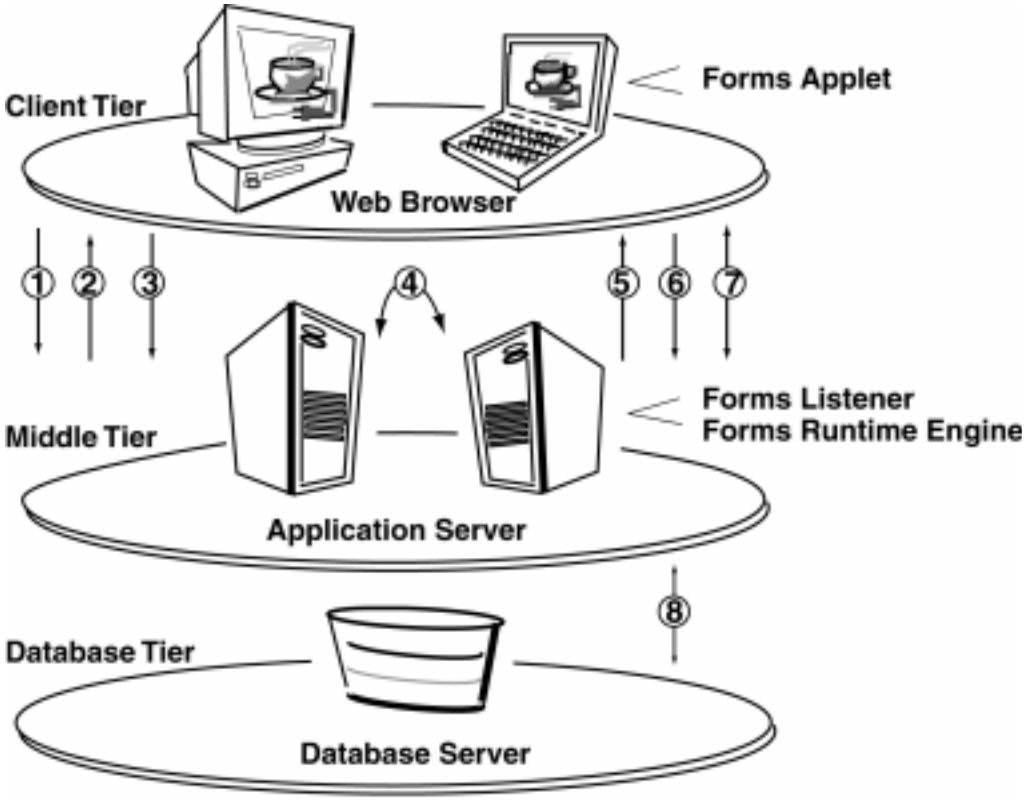


Figure 2–3 Forms Server process flow

When a user runs a Forms application on the Web, the following sequence of events occurs:

1. The user accesses the URL of an HTML page that indicates a Forms application should be run.
2. The HTML page is downloaded to the Web browser. If needed, the client will also download the Java archive file containing the Forms applet. The Forms applet will be instantiated and the parameters from the HTML page will be used to determine which Forms application will be run.
3. The Forms applet sends a request to the Forms Listener (which resides on a specific port of the machine from which the Forms applet was downloaded).
4. The Forms Listener contacts the Forms Runtime Engine and connects to a Forms Server runtime process. If included in the HTML page, Forms command-line parameters (such as form name, user ID and password, database SID, menu name, and so on) and any user-defined Form Builder parameters are passed to the process by the Forms Listener.
5. The Listener establishes a connection with the Runtime Engine, and sends the connection information to the Forms applet.
6. The Forms applet then establishes a direct connection with the Runtime Engine.
7. The Forms applet and Runtime Engine then communicate directly, freeing the Listener to accept startup requests from other users. The Forms applet displays the application's user interface in the main window of the user's Web browser.
8. The application running on the Runtime Engine communicates directly with the database.

3

Preview of Configuration Choices

3.1 Introduction

This chapter previews the choices you will face during the configuration of Forms Server and offers descriptive information to assist you in understanding the differences between options. Configuration choices include:

- Socket connection, HTTP connection, or HTTP with SSL (secure sockets layer) connection?
- Oracle JInitiator or AppletViewer?
- Load balancing or standalone configuration?
- Oracle HTTP Server or another Web server that supports CGI?

3.2 Sockets, HTTP, or HTTPS

The Forms Server can be used in three modes for deploying applications:

- Sockets
- HTTP
- HTTPS (HTTP with SSL)

Refer to Section 9.3, "Deploying Forms Server in your Network Environment" for more detailed information on the best implementation of Forms Server in your specific network environment.

3.2.1 Sockets

Like many other Internet-based technologies, Forms Server was originally designed to use sockets for communication. A Sockets is a standard programming interface to TCP / IP.

A simple way to think of sockets is to imagine a numbering system for programs that communicate over the network. Typically these programs have a client part and a server part that share a common socket number. The server listens at the common socket port for requests from the client. Communication between the client and server parts of a program are done over what is called a *socket connection*.

Here is a typical example of socket use: A client sends a request to a URL that has a non-standard port number (for example, http: / / www.xyz.com:9000). This means the client browser will attempt to connect to socket number 9000. This also means that there is a server running on www.xyz.com that listens for connections on port 9000.

The socket mode of deployment is efficient and simple to use. The Forms Server runs on a networked host machine, and it listens on a specified socket or port for connections from the Java client running on a user machine. For this method to work, the client and server machines must be able to see, or communicate with, one another directly on the network. It is not possible to use a server-side proxy in this mode.

Note: A server-side proxy is a method for keeping the machine running the server software unknown or anonymous when it is connected or providing services to the Internet. It is a security feature that is invisible to a client and used to thwart unauthorized access to the server.

If the server and the client are separated by an unsecured network, such as the Internet, socket-based deployment has potentially severe security implications.

3.2.2 HTTP

In HTTP mode, communication is also accomplished through a socket connection, but it is an HTTP socket connection. The Forms Server listens for HTTP connections from a Java client rather than for proprietary connections via sockets. All internal messaging between the Forms Server and the Java client is encapsulated in HTTP packets.

An HTTP socket connection makes it possible for sites to allow secure communication between clients and servers through a firewall. Sites that allow only HTTP traffic can deploy Forms applications through their existing firewall with little or no change to the configuration. The fact that a proxy is used is completely

transparent to the client. As far as the client knows, it has a direct connection to the Forms Server.

In the presence of a firewall, the socket mode will not work. To make a socket mode connection work through a firewall, the specific sockets or ports used by the Forms Server would have to be open and available on the firewall, which would expose your network to any traffic that locates the open socket. This essentially pierces the firewall and defeats its purpose.

HTTP is one of the most widely used protocols for deploying applications on the Internet. Organizations can lock-down their firewalls and allow only HTTP traffic, which greatly enhances the security of their private networks. Most firewall companies support the HTTP standard in their products, and many organizations are willing to allow HTTP traffic in and out of their private networks.

3.2.3 HTTPS

In HTTPS mode, communication is accomplished through an HTTP socket connection, as described in Section 3.2.2, "HTTP". However, with HTTPS, SSL (secure sockets layer) is implemented as well.

A Forms Server can use SSL as a transport protocol to provide privacy, integrity, and server authentication. SSL works at the transport level, which is one level below the application level. This means that SSL can encrypt and decrypt messages before they are handled by application-level protocols such as Telnet, FTP, and HTTP.

- **Privacy** is accomplished by encrypting messages between clients and servers, which protects messages from being read by unintended recipients. Messages are encrypted using RC4 encryption.

  Servers and clients with a domestic license support 128-bit encryption. Servers and clients with an export license support 40-bit encryption. If you have a domestic (128-bit encryption) server, clients with an export (40-bit encryption) license cannot connect unless you set the environment variable FORMS60_HTTPS_NEGOTIATE_DOWN to TRUE. (The default setting is FALSE.) See Section 5.3, "Customizing Environment Variables" for details. When you set this environment variable to TRUE, the server will always use the highest level of encryption supported by the client that is attempting to connect. If set to FALSE, clients that support encryption levels lower that the server's cannot connect. The following table shows sample implementations:

Server encryption level	Client encryption level	FORMS60_HTTPS_ NEGOTIATE_DOWN setting	Connection possible?
128-bit (domestic)	40-bit (export) 128-bit (domestic)	TRUE	Yes, 40-bit encryption for export clients and 128-bit for domestic clients
128-bit (domestic)	40-bit (export)	FALSE	No
40-bit (export)	128-bit (domestic)	TRUE	Yes, 40-bit encryption
40-bit (export)	40-bit (export)	TRUE	Yes, 40-bit encryption
40-bit (export)	40-bit (export)	FALSE	Yes, 40-bit encryption

- **Integrity** protects messages from being altered. If altered, messages cannot be decrypted correctly.
- **Server Authentication** is the process of a client machine verifying that a server is who it claims to be. For example, when a client sends confidential data to a server, the client can verify that the server is secure and is the correct recipient of the client's confidential data. Server authentication is accomplished using RSA-compliant digital certificates. When a client browser connects to a server, the server presents its certificate for verification.

If you decide to use HTTPS mode, you will need to install Oracle Wallet Manager in order to create certificate requests and manage certificates. See Section 5.6, "Additional Steps to Set Up the HTTPS Connection Mode" for details.

3.3 Oracle JInitiator or AppletViewer

Users can view applications through a browser (Netscape Navigator or Internet Explorer) combined with either the Oracle JInitiator plug-in or the AppletViewer. We recommend that you use Oracle JInitiator.

3.3.1 Oracle JInitiator

Oracle JInitiator runs within a Web browser and is the recommended method for viewing Oracle Forms applications on the Web. It provides the ability to specify the use of a specific Java Virtual Machine (JVM) on the client, rather than using the browser's default JVM. Oracle JInitiator does not replace or modify the default JVM

provided by the browser. Rather, it provides an alternative JVM in the form of a plug-in.

Oracle JInitiator is Oracle's version of JavaSoft's Plug-In. It runs as a plug-in for Netscape Navigator and as an ActiveX component for Internet Explorer.

Some configuration is required to configure and deploy Oracle JInitiator. Refer to Appendix B, "Oracle JInitiator" for more information.

3.3.2 AppletViewer

Users can also view applications using the AppletViewer. The AppletViewer is a Java Developer Kit (JDK) component that client machines use to view applications running on the Forms Server.

Running an application within the AppletViewer requires some configuration. Refer to Appendix C, "AppletViewer" for more information on running applications with the AppletViewer.

3.4 Load Balancing or standalone configuration

Forms Server includes load-balancing capabilities to optimize hardware resources for scaling from one to thousands of users with unprecedented performance. With load balancing, when you approach the limits of your hardware, rather than upgrading or replacing a machine, you simply add more machines to run your application and spread the load across several machines.

Refer to Chapter 12, "Load Balancing Considerations" for specific information on implementing load balancing.

3.5 Oracle HTTP Server or another Web server

In order to run Oracle Forms applications on the Web, a Web Server is required in addition to Forms Server. You can choose to use:

- Oracle HTTP Server
- Another Web server that supports CGI

For your convenience, Oracle HTTP Server, a Web server that supports CGI (Common Gateway Interface), is provided with the Oracle Internet Application Server installation.

You can choose not to use the Oracle HTTP Server if you already have another Web server and would prefer to use that one. Forms Server works with any Web server that supports CGI, such as Microsoft IIS or Lotus Domino. After installation is completed, you will need to configure some virtual paths with your Web server for use with Forms Server. Instructions telling you what to are located in Chapter 5.2, "Configuring Your Web Server".

3.6 What's Next

After deciding what your choices are, you can configure the necessary Forms Server components. Refer to Chapter 4, "Installing Forms Server" for information about using the Oracle Universal Installer to install the Forms Server. Refer to Chapter 5, "Configuring the Forms Server" for more information about configuring the Forms Server.

4

Installing Forms Server

4.1 Introduction

Forms Server is installed as part of the Enterprise Edition of the Oracle Internet Application Server. The Enterprise Edition is recommended for medium to large sized websites that handle a high volume of transactions.

For more detailed information about installing Forms Server, refer to the Oracle Internet Application Server Installation Guide. All necessary requrements and tasks are documented in the installation guide.

4.2 About the Oracle Universal Installer

Oracle Internet Application Server uses the Oracle Universal Installer, a Java-based tool, to configure environment variables and to install components. The installer guides you through each step of the installation process, so you can choose different configuration options.

The installer includes features that perform the following tasks:

- Explore and provide installation options for the product
- Detect pre-set environment variables and configuration settings
- Set environment variables and configuration settings during installation
- De-install the product

4.3 Starting Forms Server

After installation is completed, the Forms Server is started automatically.

To manually start the Forms Server, type:

```
$ORACLE_HOME/6iserver/ forms60_server start
```

To stop the Forms Server, type:

```
$ORACLE_HOME/6iserver/ forms60_server stop
```

4.4 What's Next

To actually deploy your applications, you must perform several steps, which include creating your runtime executable files, deploying the executable files on your Web server, and broadcasting your application's URL. These steps are described in Chapter 6, "Deploying Forms to the Web".

5

Configuring the Forms Server

5.1 Introduction

This chapter describes the steps you need to follow to configure your environment for Forms Server. After installation is complete, you can use the information in this chapter to change your initial configuration or make modifications as your needs change.

This chapter contains the following sections:

- Configuring Your Web Server
- Customizing Environment Variables
- Customizing Configuration Files
- Additional Steps to Set Up the HTTPS Connection Mode

5.2 Configuring Your Web Server

Oracle Internet Application Server installs and configures the Oracle HTTP Server as your Web server. No additional configuration is necessary.

If you choose to use another Web server, read the documentation provided to configure it for use with Forms Server. To configure another Web server to run with Forms Server, you need to create the following virtual paths:

Virtual Path	Physical Directory	Description
/forms60java/	<$ORACLE_HOME>/forms60/java/	Forms Java files
/dev60html/	<$ORACLE_HOME>/tools/web60/html/	Starter HTML files for running Forms
/dev60cgi/	<$ORACLE_HOME>/tools/web60/cgi/	CGI executables
/jinitiator/	<$ORACLE_HOME>/jinit/	JInitiator (for download)
/dev60temp/	<$ORACLE_HOME>/tools/web60/temp/	Forms temporary files

Note: These virtual directories are specified in the 6iserver.conf file located in the $ORACLE_HOME/6iserver directory.

5.3 Customizing Environment Variables

This section describes how to customize environment variables in Forms Server. You can set these environment variables in the forms60_server shell script, which is found in the $ORACLE_HOME/6iserver directory. This way, all the environment variables needed for Forms Server are automatically set up when you launch the Forms Server Listener using the following command line:

```
forms60_server start.
```

Note: After you run the forms60_server startup script, ORACLE_HOME changes from its original setting to $ORACLE_HOME/6iserver for use with Forms Server.

The environment variables for Forms Server are as follows:

Environment Variable	Default Value and Description
FORMS60_PATH	<ORACLE_HOME>/forms60 Specifies the path that Forms searches when looking for a Form to run. Separate paths with a semi-colon (;).

Environment Variable	Default Value and Description
FORMS60_OUTPUT	<ORACLE_HOME>/tools/web60/temp Physical directory on the application server in which to store generated Reports files. If you are not using Reports, this environment variable is not required. See Section 7.5, "Integrating Reports" for more information.
FORMS60_MAPPING	/dev60temp Virtual directory pointing to the physical directory defined by the FORMS60_OUTPUT variable. If you are not using Reports, this environment variable is not required. See Section 7.5, "Integrating Reports" for more information.
FORMS60_MESSAGE_ENCRYPTION	Not set Possible values are TRUE or FALSE. Environment variable to encrypt Forms messages using RC4 40-bit encryption. Applies only to socket and HTTP communication modes. By default, communication is encrypted.
FORMS60_WALLET	<ORACLE_HOME>/forms60/wallet Used for HTTPS communications mode only. See Section 5.6, "Additional Steps to Set Up the HTTPS Connection Mode" for details.
FORMS60_HTTPS_NEGOTIATE_DOWN	FALSE Used for HTTPS communications mode only. See Section 5.6, "Additional Steps to Set Up the HTTPS Connection Mode" for details.

For example, you can define your environment variables as the following:

```
FORMS60_PATH=/<ORACLE_HOME>/forms60
FORMS60_OUTPUT=/<ORACLE_HOME>/tools/web60/temp
FORMS60_MAPPING=/dev60temp
FORMS60_MESSAGE_ENCRYPTION=TRUE
FORMS60_WALLET=/<ORACLE_HOME>/forms60/wallet
FORMS60_HTTPS_NEGOTIATE_DOWN=FALSE
```

Note: The virtual directory set by the FORMS60_MAPPING environment variable *must* correspond to the physical directory set by the FORMS60_OUTPUT environment variable.

Note: You will need administrator privileges to make these changes, and will need to restart the server for many of these configuration changes to take effect.

5.4 Description of Forms Server Startup Parameters

The following parameters are used during Forms Server startup:

- Port Parameter
- Mode Parameter
- Pool Parameter
- Log Parameter

You can modify these prameters by editing the forms60_server shell script found in the $ORACLE_HOME/6iserver directory and modifying the following command:

```
f60ctl start
```

For example:

```
f60ctl start port=9001 mode=socket pool=5 log=/tmp/app.log
```

5.4.1 Port Parameter

Determines the port on which the server process is started. If you do not specify a port number when you start the Forms Server process, the process starts on port 9001 by default. The port number on which you start the server process must match the serverPort number you specify in an application's HTML file, configuration parameters, or URL.

5.4.2 Mode Parameter

Determines whether the Forms Server will run in socket mode (which uses a direct socket connection), HTTP mode (which can traverse firewalls), or HTTPS mode (which can traverse firewalls, and additionally uses SSL, secure sockets layer, for server authentication and message encryption). The default mode is socket. See Section 3.2, "Sockets, HTTP, or HTTPS" for a detailed description of each mode.

5.4.3 Pool Parameter

Determines the number of spare active connections that will be available for subsequent users. For example, if "pool" is set to 5, there will be 5 active spare connections.

5.4.4 Log Parameter

Generates a server log file when provided a path name and log file name, for example, log=/PathName/LogFileName.

5.5 Customizing Configuration Files

During the installation, the following configuration files were installed onto your system:

- formsweb.cfg
- base.htm and basejini.htm

When a user first starts a Web-enabled application (by clicking a link to the application's URL), the base HTML file is read by the Forms CGI. Any variables (%*variablename*%) in the base HTML file are replaced with the appropriate parameter values specified in the formsweb.cfg file and from query parameters in the URL request (if any).

You can modify the configuration files as your needs change. The files are located in the $ORACLE_HOME/6iserver/forms60/server directory after installation.

5.5.1 formsweb.cfg

This file contains most of the configuration parameter settings that you set during installation. You can modify these parameters, if needed.

Variables (%*variablename*%) in the base HTML file are replaced with the appropriate parameter values specified in the formsweb.cfg file and from query parameters in the URL request (if any).

Variables (%*variablename*%) can also be used in the formsweb.cfg file. (In this case, the delimiter is always %). The variables must be either Oracle registry or environment variables (such as <ORACLE_HOME>) or the special variable %leastloadedhost%.

We recommend that you enter configuration changes in the formsweb.cfg file, and use variables in the baseHTML file.

5.5.1.1 Parameters in the formsweb.cfg File

Parameter	Required / Optional	Parameter Value
baseHTML	**required**	Physical path to HTML file that contains applet tags.
baseHTMLJInitiator	**required**	Physical path to HTML file that contains JInitiator tags.
ie50	recommended if there are users with Internet Explorer 5.0 browsers	If the client is using the Internet Explorer 5.0 browser, either JInitiator or AppletViewer can be used. A setting of "JInitiator" uses the basejini.htm file and JInitiator. A setting of "Native" uses the browser's native JVM.
HTML delimiter	**required**	Delimiter for variable names. Defaults to %.
MetricsServerHost	optional	For load balancing. See Chapter 12, "Load Balancing Considerations".
MetricsServerPort	optional	For load balancing. See Chapter 12, "Load Balancing Considerations".
MetricsServerErrorURL	optional	For load balancing. See Chapter 12, "Load Balancing Considerations".
MetricsTimeout	optional	For load balancing. See Chapter 12, "Load Balancing Considerations".
leastloadedhost	optional	For load balancing. See Chapter 12, "Load Balancing Considerations". This is a variable that can be specified in either the base HTML file or the formsweb.cfg file, wherever the name of the least loaded machine is required for load balancing. If you use the default base HTML file, which is recommended, then be sure to specify serverHost=%loastloadedhost% in the formsweb.cfg file when load balancing is being used. During load balancing, this placeholder is replaced dynamically with the name of the least-loaded system.

Standard applet or object Parameters

Note: All of the following can be specified in the base HTML file as %*variablename*%. For example:

```
<PARAM NAME="connectMode"  VALUE="%connectMode%">
```

All variables in the base HTML file are replaced with the appropriate parameter values specified in the formsweb.cfg file.

Parameter	Required / Optional	Parameter Value
codebase	**required**	Virtual directory you defined to point to the physical directory <ORACLE_HOME>/forms60/java.
code	**required**	Do not remove or modify the code parameter. Its value should always be: oracle.forms.engine.Main.
connectMode	**required for HTTP and HTTPS connections**; optional for socket connection	Specifies to the client the type of connection protocol to use with the Forms Server. Valid values are socket, http, and https. The default is socket. See Section 3.2, "Sockets, HTTP, or HTTPS" for details.
archive	optional	Comma-separated list of archive files to preload. Paths, if not absolute, are relative to codebase.
width	**required**	Specifies the width of the Form, in pixels.
height	**required**	Specifies the height of the Form, in pixels.
align	optional	left | center | right | top | middle | bottom
alt	optional	Text displayed instead of applet (if browser does not support applets)
hspace	optional	Horizontal gutter, in pixels.
vspace	optional	Vertical gutter, in pixels.
type	**required**	Hard coded value ("application/x-jinit-applet" for JInitiator; no value required for AppletViewer).
name	optional	Applet instance name.
title	optional	Advisory title string.
border	optional	Border to display.
standby	optional	Text to display when loading.
codetype	optional	Defaults to type.
Parameters specific to the Forms applet (in PARAM tags)		
serverHost	optional	Host on which the Forms Server, ifsrv60.exe runs (defaults to Web listener machine).
serverPort	required	Port on which the Forms Server, ifsrv60.exe listens. In most cases, the port number will remain 9001 (the default).

Parameter	Required / Optional	Parameter Value
serverArgs	**required**	Command-line parameters for Runform. See Runform parameters below. Replace forms_param with any valid Form Runtime command-line parameter. Replace user_param with any valid user-defined parameter. For example, <param name="serverArgs" VALUE="module=order.fmx"> **Notes**: You can provide multiple Form Runtime command-line and user-defined parameters. You must provide a physical directory path for the .FMX file by including a directory path by defining the FORMS60_PATH environment variable. The .FMX suffix is optional.
splashScreen	optional	Specifies the .GIF file that should appear before the applet appears. Set to NO for no splash. Leave empty to use the default splash.
background	optional	Specifies the .GIF file that should appear in the background. Set to NO for no background. Leave empty to use the default background.
clientDPI	optional	Specifies the dots per inch (DPI) and overrides the DPI setting returned by the JVM, allowing you to manage varying DPI settings per platform. For example, a form developed on the Win32 platform may not display properly on the UNIX platform due to varying DPI values. The clientDPI value can be any positive integer. Oracle recommends that you use an integer between 50 and 200. <param name="clientDPI" value="200">
separateFrame	optional	Determines whether the applet appears within a separate frame. Legal values: True or False.
lookAndFeel	optional	Determines the applications look-and-feel. Legal values: Oracle or Generic (Windows 95 look-and-feel).
colorScheme	optional	Determines the application's color scheme. Legal values: Teal, Titanium, Red, Khaki, Blue, Olive, or Purple. **Note**: colorScheme is ignored if lookAndFeel is set to Generic.
serverApp	optional	Replace default with the name of your application class (if any). Use application classes for creating application-specific font mapping and icon path settings.
heartBeat	optional	Use this parameter to set the frequency at which a client sends a packet to the server to indicate that it is still running. Define this integer value in minutes. The default is two minutes.

Parameter	Required / Optional	Parameter Value
imageBase	optional	Use this parameter to indicate where icon files are stored. Choose between: ■ **codeBase**, which indicates that the icon search path is relative to the directory that contains the Java classes. Use this value if you store your icons in a JAR file (recommended). ■ **documentBase**, which is the default. In deployments that make use of the Forms Server CGI, you must specify the icon path in a custom application file.
registryPath	optional	Use this parameter to list the virtal directory where the application file named in the serverApp parameter is located.
webformsTitle	optional	Use this parameter to change the title that appears in the top border of a form's display window.
Runform parameters (serverArgs parameters)		
MODULE	**required**	Form module name (optionally includes path).
USERID	optional	Login string, such as scott/tiger@ORA8.
user-defined parameters	optional	Arbitrary name/value pairs.

5.5.1.2 Default formsweb.cfg File

The default formsweb.cfg file contains the following:

```
; Forms Web CGI Configuration File
; --------------------------------
; This file defines parameter values used by the Forms Web CGI
; ********************************
; PARAMETER VALUES USED BY DEFAULT
; ********************************
  ; SYSTEM PARAMETERS
  ; -----------------
  ; These have fixed names and give information required by the Forms
  ; Web CGI in order to function.  They cannot be specified in the URL query
  ; string.  But they can be overriden in a named configuration (see below).
baseHTML=<FORMS60>\server\base.htm
baseHTMLJInitiator=<FORMS60>\server\basejini.htm
HTMLdelimiter=%
MetricsServerPort=9020
```

```
MetricsServerErrorURL=
  ; The next parameter specifies how to execute the Forms applet under
  ; Microsoft Internet Explorer 5.0.  Put IE50=native if you want the
  ; Forms applet to run in the browser's native JVM.
IE50=JInitiator

  ; USER PARAMETERS
  ; ---------------
  ; These match variables (e.g. %form%) in the baseHTML file. Their values
  ; may be overridden by specifying them in the URL query string
  ; (e.g. "http://myhost.mydomain.com/ifcgi60.exe?form=myform&width=700")
  ; or by overriding them in a specific, named configuration (see below)
  ; 1) Runform arguments:
form=test.fmx
userid=
otherparams=

  ; 2) HTML page title, attributes for the BODY tag, and HTML to add before and
  ;    after the form:
pageTitle=Forms Server
HTMLbodyAttrs=
HTMLbeforeForm=
HTMLafterForm=

  ; 3) Values for the Forms applet parameters:
width=650
height=500
separateFrame=false
splashScreen=no
background=no
lookAndFeel=Oracle
colorScheme=teal
serverApp=default
serverPort=9000
serverHost=
connectMode=socket
archive=f60web.jar

  ; 4) Parameters for JInitiator
    ; Page displayed to Netscape users to allow them to download JInitiator.
    ; If you create your own version, set this parameter to point to it.
jinit_download_page=/jinitiator/us/jinit_download.htm
    ; Parameters related to the version of JInitiator.
    ; These are valid for Oracle JInitiator version 1.1.7.16o
    ; WARNING: You must update these if you upgrade to a later version
```

```
    ; of JInitiator (as instructed in the documentation for that version)
jinit_classid=clsid:9F77A997-F0F3-11d1-9195-00C04FC990DC
jinit_exename=jinit.exe#Version=1,1,7,16
jinit_mimetype=application/x-jinit-applet;version=1.1.7.16
    ; Values for JInitiator version 1.1.7.18o:
    ; jinit_classid=clsid:9F77A997-F0F3-11d1-9195-00C04FC990DC
    ; jinit_exename=jinit11718.exe#Version=1,1,7,18
    ; jinit_type=application/x-jinit-applet;version=1.1.7.18
; ******************************
; SPECIFIC CONFIGURATIONS
; ******************************
;  You may define your own specific, named configurations (sets of parameters)
;  by adding special sections as illustrated in the following examples.
;  Note that you need only specify the parameters you want to change.  The
;  default values (defined above) will be used for all other parameters.
;  Use of a specific configuration can be requested by including the text
;  "config=<your_config_name>" in the query string of the URL used to run
;  a form.  For example, to use the sepwin configuration, your could issue
;  a URL like "http://myhost.mydomain.com/ifcgi60.exe?config=sepwin".

; Example 1: configuration to run forms in a separate browser window with
;            "generic" look and feel (include "config=sepwin" in the URL)
[sepwin]
separateFrame=True
lookandfeel=Generic

; Example 2: configuration affecting users of MicroSoft Internet Explorer 5.0.
;            Forms applet will run under the browser's native JVM rather than
;            using Oracle JInitiator.
[ie50native]
IE50=native

; Example 3: configuration forcing use of the base.htm base HTML file in all
;            cases (means applet-style tags will always be generated and
;            JInitiator will never be used).
[applet]
baseHTMLJInitiator=

; Example 4: configuration to run the demos
;            PLEASE DO NOT REMOVE THIS EXAMPLE, !
;            It is needed to run the Forms demos (if they are installed)
[demo]
pageTitle=Forms Server Demos
width=700
height=550
```

```
form=start60
userid=%Demos_ConnectString%
archive=f60all.jar, oracle_ice-4_03_1.jar
serverApp=/forms60demo/demo
lookAndFeel=oracle
colorScheme=teal
```

5.5.2 base.htm and basejini.htm

Two base HTML files are created for your system by the Oracle Universal Installer during Forms Server installation and configuration. **In most cases, you will not need to modify these files.**

When a user first starts a Web-enabled application (by clicking a link to the application's URL), a base HTML file is read by the Forms CGI. Any variables (%*variablename*%) in the base HTML file are replaced with the appropriate parameter values specified in the formsweb.cfg file, described in Section 5.5.1, "formsweb.cfg" or from query parameters in the URL request (if any). Then, the base HTML file is downloaded to the user's Web browser.

Note: Any base HTML variables that you want to modify can be changed by modifying the corresponding parameter values in the formsweb.cfg file, described in Section 5.5.1, "formsweb.cfg".

The following base HTML starter files are available in the $ORACLE_HOME/6iserver/ /forms60/server directory:

- **basejini.htm**: This is a base HTML file containing the tags required to run the Forms applet using Oracle JInitiator. It is suitable for browsers (only on Windows platforms) certified by Oracle to work in this manner (and which do not work using standard APPLET tags). See Section 5.5.2.4, "Default basejini.htm File" for an example. Also, see Appendix B, "Oracle JInitiator" for more information about JInitiator settings.
- **base.htm**: This is a base HTML file containing the APPLET tags required to run the Forms applet in the AppletViewer, or in any Web browser certified by Oracle whose native JVM is certified with Forms. See Section 5.5.2.3, "Default base.htm File" for an example. Also, see Appendix C, "AppletViewer" for more information about AppletViewer settings.

If you decide to create a new base HTML file:

1. Copy the basejini.htm or base.htm starter file, which is located in the $ORACLE_HOME/6iserver/forms60/server directory.
2. Rename the file, for example, `order.htm`.

3. Add or modify any text that is visible to the user (for example text contained within <TITLE> and <BODY> tags).
4. Modify the parameters as needed. We recommend that you use variables in the base HTML file, and specify the actual values in the formsweb.cfg file, as described in Section 5.5.1, "formsweb.cfg".
5. Place the new base HTML file in any directory. Update the baseHTML parameter (or baseHTMLJInitiator parameter) in the formsweb.cfg file to contain the base HTML file's full physical path location.

5.5.2.1 Parameters and variables in the base HTML file

Note: If you do not want to use a parameter tag that is provided in the base.htm or basejini.htm file, delete it from the file.

Parameter	Required / Optional	Parameter Value
CGI system variable		
leastloadedhost	optional	For load balancing. See Chapter 12, "Load Balancing Considerations". This is a variable that can be specified in either the base HTML file or the formsweb.cfg file, wherever the name of the least loaded machine is required for load balancing. If you use the default base HTML file, which is recommended, then be sure to specify serverHost=%leastloadedhost% in the formsweb.cfg file when load balancing is being used. During load balancing, this place holder is replaced dynamically with the name of the least-loaded system.
Standard applet or object parameters **Note:** We recommend that you specify the rest of the parameter values as variables (%*variablename*%) in the base HTML file. For example: <PARAM NAME="connectMode" VALUE="*%connectMode%*"> Then, specify the actual parameter values in the formsweb.cfg file, which are defined in Section 5.5.1.1, "Parameters in the formsweb.cfg File". All variables are replaced with the appropriate parameter values at runtime.		

5.5.2.2 Usage Notes

- You can use a variable value anywhere in the base HTML file. Variables are specified as a name enclosed in a special delimiter. (The default delimiter is %.) For example, you could have the following line in your HTML file:

```
ARCHIVE="%Archive%"
```

You then must assign a value to %Archive% either in the formsweb.cfg file (or in the URL query string).

- All variables must receive values at runtime. If a variable does not receive a value, the Forms Server cannot build an HTML file to pass back to the user's Web browser, resulting in an error.
- To streamline performance, use only one Web server as a source for JAR file downloads. This will prevent multiple downloads of the same files from different servers.

5.5.2.3 Default base.htm File

```
<HTML>
<!-- FILE: base.htm (Forms Server)                          -->

<!-- This is the default base HTML file for running a form on the   -->
<!-- web using APPLET-style tags to include the Forms applet.       -->
<!-- This file will be REPLACED if you reinstall "Forms Web CGI and -->
<!-- cartridge", so you are advised to make your own version if you -->
<!-- want to make any modifications.  You should then set the       -->
<!-- baseHTML parameter in the Forms web CGI configuration file     -->
<!-- (formsweb.cfg) to point to your new file instead of this one.  -->

<!-- IMPORTANT NOTE: default values for all the variables which     -->
<!-- appear below (delimited by the percent character) are defined  -->
<!-- in the formsweb.cfg file. It is preferable to make changes in  -->
<!-- that file where possible, and leave this one untouched.        -->

<HEAD><TITLE>%pageTitle%</TITLE></HEAD>

<BODY %HTMLbodyAttrs%>
%HTMLbeforeForm%

<!-- Forms applet definition (start) -->
<APPLET CODEBASE="/forms60java/"
        CODE="oracle.forms.engine.Main"
        ARCHIVE="%archive%"
        WIDTH="%Width%"
        HEIGHT="%Height%">

<PARAM NAME="serverPort" VALUE="%serverPort%">
<PARAM NAME="serverHost" VALUE="%serverHost%">
```

```
<PARAM NAME="connectMode" VALUE="%connectMode%">
<PARAM NAME="serverArgs"
       VALUE="module=%form% userid=%userid% %otherParams%">
<PARAM NAME="separateFrame" VALUE="%separateFrame%">
<PARAM NAME="splashScreen"  VALUE="%splashScreen%">
<PARAM NAME="background"  VALUE="%background%">
<PARAM NAME="lookAndFeel"  VALUE="%lookAndFeel%">
<PARAM NAME="colorScheme"  VALUE="%colorScheme%">
<PARAM NAME="serverApp" VALUE="%serverApp%">

</APPLET>
<!-- Forms applet definition (end) -->

%HTMLafterForm%

</BODY>
</HTML>
```

5.5.2.4 Default basejini.htm File

```
<HTML>
<!-- FILE: basejini.htm (Forms Server)                                -->

<!-- This is the default base HTML file for running a form on the   -->
<!-- web using JInitiator-style tags to include the Forms applet.   -->
<!-- This file will be REPLACED if you reinstall "Forms Web CGI and -->
<!-- cartridge", so you are advised to make your own version if you -->
<!-- want to make any modifications.  You should then set the       -->
<!-- baseHTML parameter in the Forms web CGI configuration file     -->
<!-- (formsweb.cfg) to point to your new file instead of this one.  -->

<!-- IMPORTANT NOTE: default values for all the variables which     -->
<!-- appear below (delimited by the percent character) are defined  -->
<!-- in the formsweb.cfg file. It is preferable to make changes in  -->
<!-- that file where possible, and leave this one untouched.        -->

<HEAD><TITLE>%pageTitle%</TITLE></HEAD>

<BODY %HTMLbodyAttrs%>
%HTMLbeforeForm%

<!-- Forms applet definition (start) -->
<OBJECT classid="%jinit_classid%"
        codebase="/jinitiator/%jinit_exename%"
        WIDTH="%width%"
```

```
        HEIGHT="%Height%"
        HSPACE="0"
        VSPACE="0">
<PARAM NAME="TYPE"       VALUE="%jinit_mimetype%">
<PARAM NAME="CODEBASE"   VALUE="/forms60java/">
<PARAM NAME="CODE"       VALUE="oracle.forms.engine.Main" >
<PARAM NAME="ARCHIVE"    VALUE="%archive%" >

<PARAM NAME="serverPort" VALUE="%serverPort%">
<PARAM NAME="serverHost" VALUE="%serverHost%">
<PARAM NAME="connectMode" VALUE="%connectMode%">
<PARAM NAME="serverArgs"
       VALUE="module=%form% userid=%userid% %otherParams%">
<PARAM NAME="separateFrame" VALUE="%separateFrame%">
<PARAM NAME="splashScreen"  VALUE="%splashScreen%">
<PARAM NAME="background"  VALUE="%background%">
<PARAM NAME="lookAndFeel"  VALUE="%lookAndFeel%">
<PARAM NAME="colorScheme"  VALUE="%colorScheme%">
<PARAM NAME="serverApp" VALUE="%serverApp%">
<COMMENT>
<EMBED SRC="" PLUGINSPAGE="%jinit_download_page%"
        TYPE="%jinit_mimetype%"
        java_codebase="/forms60java/"
        java_code="oracle.forms.engine.Main"
        java_archive="%archive%"
        WIDTH="%Width%"
        HEIGHT="%Height%"
        HSPACE="0"
        VSPACE="0"

        serverPort="%serverPort%"
        serverHost="%serverHost%"
        connectMode="%connectMode%"
        serverArgs="module=%form% userid=%userid% %otherparams%"
        separateFrame="%separateFrame%"
        splashScreen="%splashScreen%"
        background="%background%"
        lookAndFeel="%lookAndFeel%"
        colorScheme="%colorScheme%"
        serverApp="%serverApp%"
>
<NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>
```

```
<!-- Forms applet definition (end) -->

%HTMLafterForm%
</BODY>
</HTML>
```

5.6 Additional Steps to Set Up the HTTPS Connection Mode

The HTTPS connection mode uses HTTP for communications in order to traverse firewalls. In addition, a Forms Server uses SSL as a transport protocol to provide privacy, integrity, and server authentication. See Section 3.2.3, "HTTPS" for a description of this communications mode.

To use the HTTPS connection mode, you must do the following before starting a Forms Server in HTTPS mode:

- Customize HTTPS Environment Variables
- Use Oracle Wallet Manager to Create Wallets and Request Certificates

Note: Oracle Wallet Manager must be installed to use the HTTPS connection mode and on all Forms Server machines that will provide server authentication.

5.6.1 Customize HTTPS Environment Variables

Two environment variables associated with HTTPS mode are set during Forms Server installation. Check that these environment variables are set to meet your security needs, and change them, if needed, on all Forms Server machines running in HTTPS mode. See Section 5.3, "Customizing Environment Variables" for information on how to change environment variables.

Environment Variable	Value
FORMS60_HTTPS_NEGOTIATE_DOWN	The default value is FALSE. Valid values are TRUE and FALSE. If set to TRUE, a server that uses 128-bit encryption will negotiate encryption down to the highest level supported by the client. If FALSE, the server will reject client connections that do not support 128-bit encryption. See Section 3.2.3, "HTTPS" for details.
FORMS60_WALLET	The default value is /<ORACLE_HOME>/forms60/wallet Directory containing the "wallet" that holds the certificate used for server authentication.

5.6.2 Use Oracle Wallet Manager to Create Wallets and Request Certificates

Public-key cryptography requires, among other things, certificates. A user certificate is issued by a third party, called a *certificate authority* (CA). The certificate is obtained in a secure manner and does not need to be validated for its authenticity each time it is accessed.

In the case of a Forms Server and Java client using HTTPS mode, the Java client uses a user certificate to validate that a Forms Server is who it claims to be by verifying the server's certificate. You use Oracle Wallet Manager to create wallets and request user certificates.

After installing Oracle Wallet Manager, you must do the following to obtain a user certificate, which is required when using the HTTPS communication mode:

- Create a Wallet.
- Create a Certificate Request.
- Import the User Certificate.
- Set Auto Login to ON.

The following sections provide an overview of how to complete the above steps in Oracle Wallet Manager. See the Oracle Wallet Manager documentation for details.

Note: If you have multiple Forms Server machines, you can request a unique certificate for each machine, or you can use the same certificate on all machines.

- **To use a unique certificate on each machine**, perform all of the procedures in this section on each Forms Server machine running in HTTPS mode.
- **To use the same certificate on all machines**, perform all of the procedures in this section on one of the Forms Server machines to create a wallet that contains a certificate. Then, copy the wallet file, ewallet.dev, to the other Forms Server machines running in HTTPS mode. Copy the file to the directory specified in the FORMS60_WALLET environment variable. Finally, be sure that Auto Login is set to ON on all machines, as described in Set Auto Login to ON.
- **To launch Oracle Wallet Manager on UNIX**, type the following: `owm`

5.6.2.1 Create a Wallet

Create a wallet as follows:

1. Click **Wallet → New** from the menu bar. The New Wallet dialog box is displayed.
2. Type a password in the Wallet Password field.

3. Retype that password in the Confirm Password field.
4. Click **OK** to continue. A message appears, and informs you that a new empty wallet has been created, and prompts you to decide whether you want to create a certificate request.
5. Click **Yes**, and see Section 5.6.2.2, "Create a Certificate Request".

5.6.2.2 Create a Certificate Request

Create a certificate request as follows:

1. Type the following information in the Certificate Request dialog box:
   - **Common Name**: Type the name of the certificate identity in First name Last name format.
   - **Organizational Unit**: Type the name of the identity's organizational unit, for example, Finance.
   - **Organization**: Type the name of the identity's organization, for example, XYZ Corp.
   - **Locality/City**: Type a city or locality.
   - **State/Province**: Type a state or province.
   - **Country**: Click the drop down list to view a list of country abbreviations. Click to select the country in which the organization is located.
   - **Key Size**: Click the drop down box to view a list of key sizes to use when creating the public/private key pair.
   - **Advanced**: Click Advanced to view the Advanced Certificate Request dialog panel. Use this field to edit or customize the identity's distinguished name (DN).
2. Click **OK**. An Oracle Wallet Manager message box informs you that a certificate request was successfully created.
3. Copy the certificate request text from the body of the message box, and paste it into an e-mail message. Send the request to a certificate authority.
4. Click **OK**. You are returned to the Oracle Wallet Manager main window. The status of the certificate is changed to Requested.

5.6.2.3 Import the User Certificate

After you receive the user certificate that you requested from the CA, you must import it into the wallet that you created. You can import it in one of two ways:

- Paste the user certificate from an e-mail that you receive from the certificate authority.
- Import the user certificate from a file.

To paste the user certificate:

1. From the menu bar, click **Operations → Import User Certificate**. The Import User Certificate dialog box opens.
2. Click the **Paste the Certificate** radio button, and click **OK**. An Import User Certificate dialog box opens with the following message: "Please provide a base64 format certificate and paste it below".
3. Copy the user certificate from the body of the e-mail you received.
4. Paste the certificate into the window, and click **OK**. A message at the bottom of the window informs you that the user certificate was successfully installed.
5. Click **OK**. You are returned to the Oracle Wallet Manager main panel, and the user certificate is displayed at the bottom of the User Certificates tree.

To import a file that contains the user certificate:

1. From the menu bar, click **Operations → Import User Certificate**. The Import User Certificate dialog box opens.
2. Type the path or folder name of the user certificate location.
3. Click to select the name of the user certificate file, for example, cert.txt.
4. Click **OK**. A message at the bottom of the window informs you that the user certificate was successfully imported into the wallet.
5. Click **OK** to close the dialog box. You are returned to the Oracle Wallet Manager main panel, and the user certificate is displayed at the bottom of the User Certificates tree.

5.6.2.4 Set Auto Login to ON

The Oracle Wallet Manager Auto Login feature automatically opens a copy of the wallet. This allows server authentication to occur without having to provide a password for the wallet.

To set Auto Login to ON:

1. Click **Wallet** from the menu bar.
2. Click the check box next to the **Auto Login** menu item.
3. A message at the bottom of the window displays "Autologin enabled".

Note: The check box next to the **Auto Login** menu item can be toggled on and off. Click the check box again to clear the check mark. This will disable autologin.

Note: Auto Login must be set to ON for all Forms Server machines that will provide server authentication.

5.7 What's Next

After completing the configuration of the Forms Server, you can deploy your applications to the Web. Refer to Chapter 6, "Deploying Forms to the Web" for more detailed information.

6

Deploying Forms to the Web

6.1 Introduction

This chapter contains information about deploying Oracle Forms applications to the Web. After you have configured the Forms Server, you can deploy your executable files and broadcast your application's URL. For information about configuring the Forms Server, see Chapter 5, "Configuring the Forms Server".

6.2 Deploying a Forms Application

To deploy a Forms application, take these steps:

- Create your runtime executable files.
- Deploy the executable files on your Web server.
- Broadcast your application's URL.

6.2.1 Creating your Runtime Executable Files

You must create the .FMX runtime executable files on the same platform as the application server on which you will deploy them.

For example, if your application server's operating system is Sun Solaris, you must use the Solaris version of the Forms Compiler component to create the .FMX files for deployment on the Web.

To compile .FMX files for the Sun Solaris operating system, use the following f60genm command line:

```
f60genm module=mymodule.fmb userid=scott/tiger
```

For more information about the forms compiler options, refer to the online help.

6.2.2 Deploying the Executable Files on Your Web Server

You can deploy your Forms applications in any directory on your Web server. You must include the appropriate directory path and filename in the base HTML file. This is the file that users access to run the application. If you created a virtual directory (e.g., /dev60html) to specify the location of your base HTML file, you should deploy your base HTML file in this location.

6.2.3 Broadcasting the Application's URL

To broadcast an application's URL, all you need to do is let your intended users know what it is. Users can contact the URL with their Java-enabled Web browser and run the corresponding application. If you created an HTML page for your application, then the URL you give to users should simply point to that page.

For example, to announce the availability of its new Order Tracking application, ABC Corp. might broadcast the following URL:

http://www.abc.com:80/appshtml/order.html

ABC's URL consists of the following components:

- **Protocol:** http
- **Domain:** www.abc.com
- **Web server listener port:** 80 (implicit)
- **HTML files virtual directory:** /appshtml
- **Static HTML file:** order.html

6.3 What's Next

After you deploy your executable files on the Web server and broadcast the application's URL, you will want to test and optimize your applications from within a Web browser.

Refer to Chapter 7, "Application Design Considerations" for guidelines and tips on designing Forms applications for Web deployment.

Refer to Chapter 11, "Performance Tuning Considerations" for more information about tuning considerations when you deploy an application over the Internet or other network environment using the Forms Server.

7

Application Design Considerations

7.1 Introduction

This chapter contains guidelines and tips for designing Forms applications for Web deployment. It includes the following sections:

- General Guidelines
- Guidelines for Designing Forms Applications
- Deploying Icons and Images Used by Forms Server
- Integrating Reports
- Feature Restrictions for Forms Applications on the Web

7.2 General Guidelines

Here are some general guidelines for designing applications for Web deployment:

- Seriously consider network factors that affect the performance of your Web applications (such as interaction with security firewalls, heavy user loads, and frequent network roundtrips to application and database servers).
- Limit the number of image items and background images you include in your forms and reports. Each time an image is required, it must download from the application server.
- Optimize your network connections where possible.
- Design your queries to execute as efficiently as possible, and ensure PL/SQL program units are compiled.

7.3 Guidelines for Designing Forms Applications

Here are some tips for designing Forms applications for Web deployment. They are discussed in greater detail in the following sections:

- Create Your Own Template HTML Files.
- Create an HTML Application Menu.
- Use Oracle Designer with the Forms Server.
- Reduce Network Traffic.
- Avoid Unnecessary Graphics and Images.
- Select Standard Fonts.

7.3.1 Create Your Own Template HTML Files

Consider creating your own HTML file templates (by modifying the templates provided by Oracle). By doing this, you can hard-code standard Forms Client applet parameters and parameter values into the template. Your template can include standard text, a browser window title, or images (such as a company logo) that would appear on the first Web page users see when they run Web-enabled forms. Adding standard parameters, values, and additional text or images reduces the amount of work required to customize the template for a specific application. To add text, images, or a window title, simply include the appropriate tags in the template HTML file.

7.3.2 Create an HTML Application Menu

As you deploy additional applications on the Web, try creating a single HTML page to serve as a centralized menu for your various Web-enabled applications. This approach eliminates the need to broadcast the URL of every application you deploy or remove. As you change your roster of available applications, simply modify the collection of links on the Web menu. Users then contact the menu URL and select from the list of available applications.

7.3.3 Use Oracle Designer with the Forms Server

Forms Server supports forms generated by Oracle Designer (32-bit, Release 1.3.2 or higher). If you use the standard Oracle Designer forms generator templates (ofg4pc1t.fmb and ofg4pc2t.fmb) to generate form and menu definitions, you can use the Forms Server to compile .FMX and .MMX files and immediately run the applications on the Web.

7.3.4 Reduce Network Traffic

To cut down on the number of network roundtrips required for users to operate your Form Builder applications on the Web, consider reducing or eliminating the following Form Builder features in your applications:

- **Mouse triggers.** Including When-Mouse-Click, When-Mouse-DoubleClick, When-Mouse-Down, and When-Mouse-Up triggers in your forms will impact speed and performance. The Forms Client must communicate with the Forms Server (necessitating a network roundtrip) each time one of these trigger fires. The When-Mouse-Move trigger is not supported due to the high number of network roundtrips required each time it fires.
- **Timers.** If your form includes a timer that fires every 1/100th of a second, users face the performance ramifications of 60,000 network roundtrips every minute. Either reduce the number of timers in your forms, or change the timing interval on which your timers fire.

7.3.5 Avoid Unnecessary Graphics and Images

Wherever possible, reduce the number of image items and background images displayed in your applications. Each time an image is displayed to application users, the image must be downloaded from the application server to the user's Web browser.

To display a company logo with your Web application, include the image in the HTML file that downloads at application startup. Do this instead of including it as a background image in the application. As a background image it must be retrieved from the database or filesystem and downloaded repeatedly to users' machines.

7.3.6 Select Standard Fonts

Most fonts are not supported across all platforms. For example, Sans Serif is a commonly-used font in Microsoft Windows applications; however, Sans Serif is not available in UNIX. When a font is not available on a platform, Form Builder attempts to use a similar font. As a result, when designing forms to deploy on the Web, be sure to follow the font guidelines listed below.

At runtime, the Forms Server maps a form's fonts into their Java equivalents. Java then renders the font in a font pre-defined for the deployment platform. To convert your form's fonts into Java equivalents, Java uses an alias list, located in the file called Registry.dat.

The following table lists the Java fonts and their equivalents on the major deployment platforms:

Table 7–1

Java Font	Windows Font	X Windows Font	Macintosh Font
Courier	Courier New	adobe-courier	Courier
Dialog	MS Sans Serif	b&h-lucida	Geneva
DialogInput	MS Sans Serif	b&h-lucidatypewriter	Geneva
Helvetica	Arial	adobe-helvetica	Helvetica
Symbol	WingDings	itc-zapfdingbats	Symbol
TimesRoman	Times New Roman	adobe-times	Times Roman

If a font from your form does not map to a Java font (through the Form Builder font alias table), Java automatically assigns a Java font to the unmapped application font.

7.4 Deploying Icons and Images Used by Forms Server

This section explains how to specify the default location and search paths for icons and images.

7.4.1 Icons

When deploying a Forms application on the Web, the icon files in ICO format (specified for an iconic button, a menu, or a window) are not used. The only file formats accessible through the Web are GIF or JPG files (GIF is the default format).

By default, the icons are found relative to the DocumentBase Directory, which is the directory containing the HTML file. If you want to store your icons in another location, you have to create an application file to specify the virtual directory where the icon files reside and the file format they use (GIF or JPG). This application file must be referenced in the HTML file.

To create a custom application file:

1. Copy the registry.dat text file found in the <ORACLE_HOME>/forms60/java/oracle/forms/registry directory to another directory. This directory must be mapped to a virtual directory for your Web server (/appfile, for example).
2. Rename this new file (myapp.dat, for example).

3. Modify the iconpath parameter specifying your icon location:

   default.icons.iconpath=/mydir or http://myhost.com/mydir
   (for an absolute path)

   or

   default.icons.iconpath=mydir
   (for a relative path, starting from the DocumentBase Directory)

4. Modify the iconextension parameter:

   default.icons.iconextension=gif

   or

   default.icons.iconextension=jpg

To reference the application file in the HTML file:

In the formsweb.cfg file or your HTML file, modify the value of the serverApp parameter and set the value to the location and name of your application file.

```
<PARAM NAME="serverApp" VALUE="/appfile/myapp">
```
(for an absolute path)

or

```
<PARAM NAME="serverApp" VALUE="appfile/myapp">
```
(for a relative path, relative to the CodeBase directory)

7.4.2 SplashScreen and Background Images

When you deploy your applications to the Web, you have the ability to specify a splash screen image (displayed during the connection) and a background image file.

Those images are defined in the HTML file or in the formsweb.cfg file:

```
<PARAM NAME="splashScreen" VALUE="splash.gif">

<PARAM NAME="background" VALUE="back.gif">
```

The default location for the splash screen and background image files is in the DocumentBase directory containing the base HTML file.

7.4.3 Using a Custom JAR File Containing Icons and Images

Each time you use an icon or an image (for a splash screen or background), an HTTP request is sent to the Web server. To reduce the HTTP roundtrips between the client and the server, you have the ability to store your icons and images in a Java archive (JAR) file. Using this technique, only one HTTP roundtrip is necessary to download the JAR file.

7.4.3.1 Creating a JAR File

The SunSoft JDK comes with an executable called *jar*. This utility enables you to store files inside a Java archive. See www.java.sun.com for further information.

For example:

```
jar -cvf myjar.jar Splash.gif Back.gif icon1.gif
```

This command store three files (Splash.gif, Back.gif, icon1.gif) in a single JAR file called myjar.jar.

7.4.3.2 Using Files Within the JAR File

The default search path for the icons and images is relative to the DocumentBase. However, when you want to use a JAR file to store those files, the search path must be relative to the CodeBase directory, the directory which contains the Java applet.

If you want to use a JAR file to store icons and images, you must specify that the search path is relative to CodeBase using the imageBase parameter in the base HTML file.

This parameter accepts two different values:

- **DocumentBase** The search path is relative to the DocumentBase directory. It is the default behavior.
- **CodeBase** The search path is relative to the CodeBase directory, which gives the ability to use JAR files.

In this example, we use a JAR file containing the icons and we specify that the search should be relative to CodeBase. If the parameter "imageBase" is not set, the search is relative to DocumentBase and the icons are not retrieved from the JAR file.

For example:

<PARAM NAME="archive" VALUE="icons.jar">

<PARAM NAME="imageBase" VALUE="CodeBase">

7.4.4 Search Path for Icons and Images

The icons and images search path depends on:

- What you specify in your custom application file (for the icons)
- What you specified in the SplashScreen and Background parameters of your HTML file (for the images)
- What you specify in the imageBase parameter in your HTML file (for both icons and images)

Forms Server searches for the icons depending on what you specify. This example assumes :

- *host* is the host name.
- *documentbase* is the URL pointing to the HTML file.
- *codebase* is the URL pointing to the location of the starting class file (as specified in the HTML file).
- *mydir* is the URL pointing to your icons or images directory.

7.4.4.1 DocumentBase

The default search path is relative to the DocumentBase. In this case, you do not need to specify the imageBase parameter:

Table 7–2

	Location specified	Search path used by Forms Server
Icons	default	http://host/documentbase
	iconpath=mydir (specified in your application file)	http://host/documentbase/mydir (relative path)
	iconpath=/mydir (specified in your application file)	http://host/mydir (absolute path)
Images	file.gif (specified in your HTML file)	http://host/documentbase/file.gif

Table 7–2

	Location specified	Search path used by Forms Server
	mydir/file.gif (specified in your HTML file)	http://host/documentbase/mydir/file.gif (relative path)
	/mydir/file.gif (specified in your HTML file)	http://host/mydir/file.gif (absolute path)

7.4.4.2 CodeBase

Use the imageBase=CodeBase parameter in the base HTML file to enable the search of the icons and images in a JAR file:

Table 7–3

	Location specified	Search path used by Forms Server
Icons	default	http://host/codebase or root of the JAR file
	iconpath=mydir (specified in your application file)	http://host/codebase/mydir or in the mydir directory in the JAR file (relative path)
	iconpath=/mydir (specified in your application file)	http://host/mydir (absolute path) No JAR file is used
Images	file.gif (specified in your HTML file)	http://host/codebase/file.gif or root of the JAR file
	mydir/file.gif (specified in your HTML file)	http://host/codebase/mydir/file.gif or in the mydir directory in the JAR file (relative path)
	/mydir/file.gif (specified in your HTML file)	http://host/mydir/file.gif (absolute path) No JAR file is used.

7.5 Integrating Reports

To invoke Reports from a Web-enabled form, use the RUN_PRODUCT built-in subprogram.

To use RUN_PRODUCT to run a report from a form running on the Web, you must set three environment variables:

Table 7–4

Environment Variable	Description
FORMS60_OUTPUT	Physical directory on the application server in which to store generated Reports files. For example: $ORACLE_HOME/tools/web60/temp
FORMS60_MAPPING	Virtual directory pointing to the physical directory defined by the FORMS60_OUTPUT variable. For example: /dev60temp/
FORMS60_REPFORMAT	Format in which to store generated Reports output. For example: PDF or HTML

On Windows NT, you define your environment variables in the Registry. On UNIX, you define your environment variables in the command shell. For more information on setting up environment variables, refer to Appendix C, "AppletViewer".

After you set the environment variables above, the following sequence occurs automatically when a form running on the Web calls RUN_PRODUCT to invoke Reports.

If the output format of the report is SCREEN or PREVIEW:

- The resulting output is stored (as a temporary file with an auto-generated filename) in the physical directory specified by the FORMS60_OUTPUT environment variable.
- The Web server looks for the temporary filename (in the virtual directory defined by the FORMS60_MAPPING environment variable).
- The Web server checks the desired display format specified by the FORMS60_REPFORMAT environment variable, and displays the report in that format in the user's browser.

If the output format of the report is FILE:

- The report does not display in the user's browser.

- The resulting file is stored in the physical directory specified by the FORMS60_OUTPUT environment variable.
- The filename of the report file is the same name that is defined in the form definition.

7.6 Feature Restrictions for Forms Applications on the Web

When designing forms for eventual deployment on the Web, keep in mind that certain Forms features behave differently—or not at all—when a form is deployed on the Web. Table 7–5 lists Forms features, whether the feature is supported on the Web, and any guidelines or notes about the feature.

Table 7–5

Feature	Support	Guidelines and Notes
ActiveX, OCX, OLE, VBX	No	Third-party controls that display screen output on the application server are not supported because users cannot view the output.
When-Mouse-Enter / Leave / Move triggers	No	Each execution of the trigger requires a network roundtrip, which would downgrade performance.
console	Yes	To display the console (includes the status and message lines) to users, set the form-level property Console Window to the window in which you wish to display the console.
firewall	Yes	You must run Forms Server in HTTP or HTTPS mode and have a firewall supporting HTTP 1.1. protocol.
HOST_COMMAND, ORA_FFI, USER_EXIT	Yes	Calls to these functions often display visual output or GUI elements on users' machines in client/server mode. In a Web implementation, the same calls will display the output and GUI elements on the application server (where users cannot see or interact with them).
iconic buttons	Yes	Icon image files must be in GIF format (and not in ICO format).
NLS, BIDI	Yes	Supported for 8-bit languages only.

8

Migrating Legacy Applications to the Web

8.1 Introduction

If you are currently using the client/server version of Forms Server, migrating applications to Forms Server for the Web is straightforward. This chapter briefly describes the differences between client/server and Web implementations, and then gives guidelines to migrate your current applications from client/server-based to Web-based Forms Server.

Traditionally, load balancing services in Oracle Forms Server were supplied via an Oracle Application Server (OAS) cartridge. If you wanted to deploy forms on the Web via a Common Gateway Interface (CGI) implementation in lieu of an OAS cartridge, load balancing was not an option.

With the release of Oracle Forms Server 6i, you can now use load balancing with forms applications that are deployed on the Web via a CGI implementation. With load balancing, when you approach the usage limits of your hardware, rather than upgrading or replacing a machine, you can simply add more machines to run your application and balance the load of server traffic across several machines.

If you have already deployed Web-based Forms Developer applications via an OAS cartridge and you wish to switch to CGI, you will need to install Oracle Forms Server 6i and configure it for CGI. If you have already installed Oracle Forms Server 6i, you'll need to reconfigure it for CGI.

The purpose of this chapter is to help those with existing cartridge-based implementations install or reconfigure Oracle Forms Server 6i from OAS cartridges to non-OAS CGI.

8.1.1 Client/Server-Based Architecture

In the client/server-based implementation, shown in Figure 8–1, the Forms Server Runtime Engine and all application logic are installed on the user's desktop machine. All user interface and trigger processing occurs on the client, except for database-server-side triggers and logic that may be included in some applications.

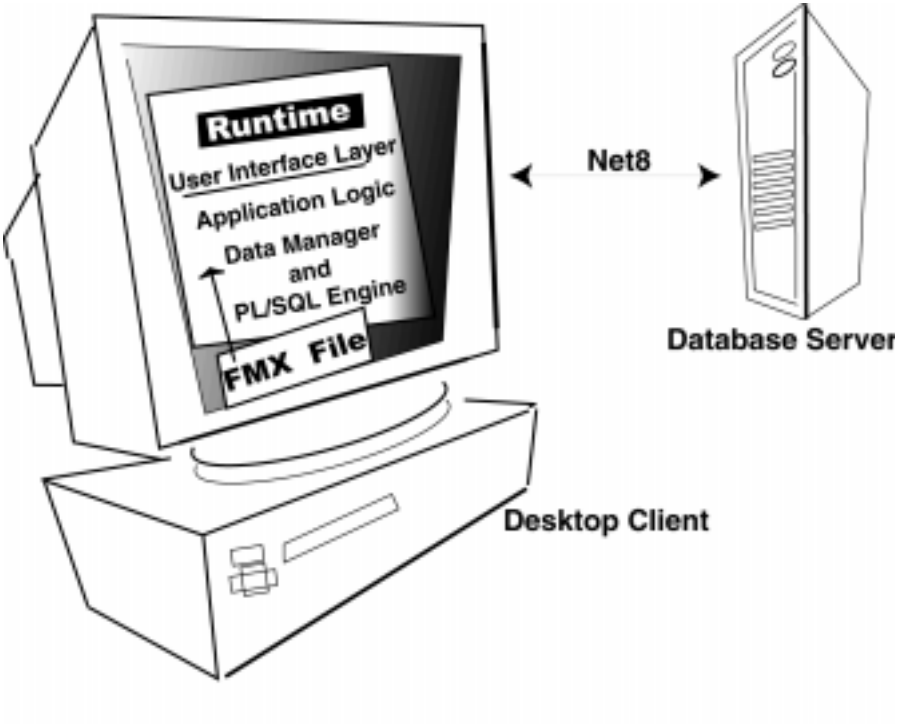


Figure 8–1 Forms Server client/server-based architecture

8.1.2 Web-Based Architecture

In a Web-based implementation, shown in Figure 8–2, the Forms Server Runtime Engine and all application logic are installed on application servers, and not on client machines. All trigger processing occurs on the database and application servers, while user interface processing occurs on the Forms client, located on users' machines.

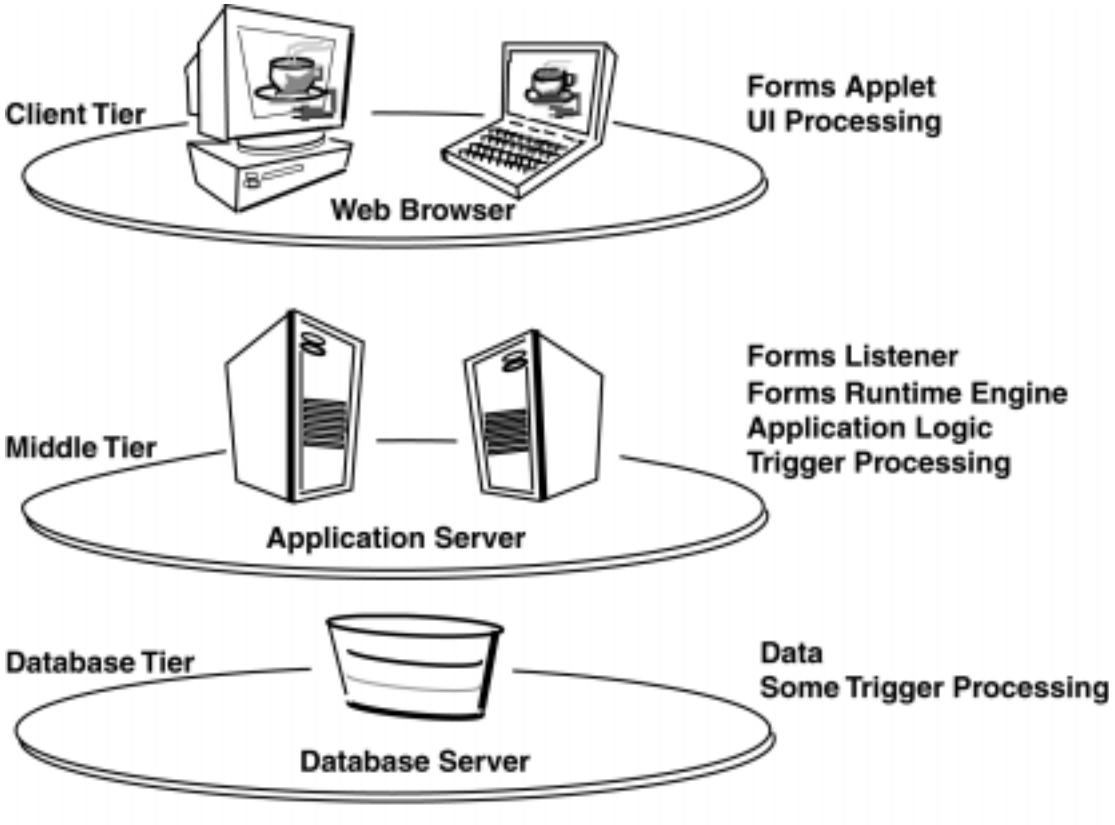


Figure 8–2 Forms Server Web-based architecture

8.1.3 Who Should Read this Chapter?

This chapter will be useful to you if the following statements apply to your deployment environment:

- You currently deploy Web-based Oracle Forms Developer applications.
- You use OAS for Web server support.
- You deploy Web-based Oracle Forms Developer applications using OAS cartridges.
- You want to move from cartridge deployment to CGI.

8.2 Comparing Cartridge and CGI Implementations

Cartridge and CGI implementations both require that you set server operational parameters that define values for such things as port numbers and locations of relevant files. The difference is in where you set them. In OAS, you open the OAS Manager and navigate to various destinations to set parameters for different deployment entities. In Oracle Forms Server, configuration complexity is more centralized. You set many operational parameters automatically through configuration choices you make during installation. You can revise and set additional operational parameters in Oracle Forms Server's formsweb.cfg file, which is created during installation.

Cartridge and CGI implementations both produce an HTML file on-the-fly that is rooted in a standard base HTML file. In a cartridge implementation, the HTML file is created through a combination of the cartridg.html file, cartridge configuration settings, and the application's URL. In a Forms Web CGI implementation, the HTML file is created through a combination of the base.htm or basejini.htm file, the formsweb.cfg file, and the application's URL.

In both cartridge and CGI base HTML files, you can define a parameter with a variable and then define the variable value in the application's cartridge settings (OAS), the formsweb.cfg file (Oracle Forms Server 6i), or via a query string in the application's URL (both OAS and Oracle Forms Server).

The major differences between cartridge and CGI implementations are in the types of services and level of performance offered through your non-OAS Web server (as compared to those offered through OAS), the broader range of operational parameters now available through Oracle Forms Server 6i, and the vastly simplified process of setting forms parameters via Oracle Forms Server installation and the formsweb.cfg file.

8.3 Reconfiguration Strategies

This section provides a high-level overview of the reconfiguration process. It is suitable for users who have a technical understanding of OAS, Oracle Forms Server, base HTML files, and the like.

There are two basic strategies for reconfiguring cartridge deployments to CGI:

- Keep everything the same, replicating cartridge parameters in the formsweb.cfg file.
- Use the default Oracle Internet Application Server installation.

The first strategy is appropriate for users with complex base HTML files, that is, files that contain much extraneous text, images, and other objects in addition to the Forms applet tags. The second strategy is appropriate for users with simple base HTML files.

8.3.1 Strategy for Users with Complex Base HTML Files

The strategy for users with complex base HTML files is to keep everything the same.

1. Stop all instances of OAS you will no longer use.
2. Install Oracle Internet Application Server.
3. In the formsweb.cfg file, reproduce the parameters that were used for cartridge configuration.

   To locate current OAS cartridge parameters, launch OAS and navigate to each forms application's Cartridge Configuration folder. Within each folder, click Cartridge Parameters. This displays the OAS cartridge parameter settings. Additionally, you will find parameter settings in the base HTML file(s) you created for Forms cartridge applications.

4. If you were using several OAS cartridge definitions for the Forms cartridge (that is, you were using several base HTML files), define separate configuration sections in the formsweb.cfg file-one for each cartridge. (With both strategies, the better practice is to create new configuration sections for cartridge parameters in the formsweb.cfg file rather than specify the parameters at the start of the formsweb.cfg file, outside a named section.)
5. For a non-OAS Web listener, define the same virtual paths you used with OAS. Add a new virtual path for CGI scripts that points to the directory containing the Forms CGI (ifcgi60.exe on NT, f60cgi on UNIX), as follows:

virtual_path_name = /dev60cgi/
NT: physical_path = %ORACLE_HOME%\tools\web60\cgi
UNIX: physical_path = $ORACLE_HOME/tools/web60/cgi

6. Change all the URL's you use to run forms to point to the CGI rather than the cartridge. For example, if the original URL was:

   ```
   http://servername.my.domain.com/developerforms/forms60cart?module=emp.fmx
   ```

   It should become:
   NT:
   ```
   http://servername.my.domain.com/dev60cgi/ifcgi60.exe?config=myconfig&
   module=emp.fmx
   ```

   UNIX:
   ```
   http://servername.my.domain.com/dev60cgi/f60cgi?config=myconfig&
   module=emp.fmx
   ```

   In this example, "myconfig" is the name of the configuration section you defined in the formsweb.cfg file that contains the parameters equivalent to your old cartridge parameters.

8.3.2 Strategy for Users with Simple Base HTML Files

The strategy for users with simple base HTML Files is to use the default Oracle Internet Application Server installation.

If your cartridge implementation used simple base HTML files, your reconfiguration to CGI can easily benefit from the default configuration that is created automatically during the installation.

1. Stop all OAS instances you will no longer use.
2. Install Oracle Internet Application Server.
3. Adapt the URLs you used to run your forms to achieve the same effect as you had with the cartridge. Use the parameters that are defined for you in the formsweb.cfg file. These allow you to change just about every conceivable HTML and Forms Applet parameter value by specifying the value in the application's URL. The same URL will work for users of the AppletViewer, a Web browser in combination with Oracle JInitiator, or Internet Explorer 5.0.
4. Use the runform.htm file to experiment with different parameter settings in the application's URL. For example, this might be the URL you would use to run a form with a page title "My Form," a page width of 400, and a page height of 550:

NT:
```
http://servername.my.domain.com/dev60cgi/ifcgi60.exe?pagetitle=
My+Form&width=400&height=550
```

UNIX:
```
http://servername.my.domain.com/dev60cgi/f60cgi?pagetitle=
My+Form&width=400&height=550
```

In these examples, the question mark signals the start of the query string in the application URL. The query string specifies the values for the pagetitle, width, and height parameters.

8.4 Reconfiguring Forms Web Cartridge to CGI

Take these steps to reconfigure forms deployments from OAS cartridge to CGI:

1. Stop the OAS Web Listener instances you will no longer use.
2. Install Oracle Internet Application Server.
3. Configure the Oracle Forms Server formsweb.cfg file.
4. Optionally, configure the Oracle Forms Server base.htm and basejini.htm files.
5. Broadcast the application's URL.

8.4.1 Stopping OAS Web Listener Instances

There are two scenarios for stopping OAS:

- Stop it completely.
- Stop only specific instances while OAS continues to support other instances.

8.4.1.1 Stopping OAS Completely

Use this technique if you wish to stop using all services offered through OAS:

1. Launch OAS.
2. Open the OAS Manager.
3. Navigate to the top-level site of the OAS installation.
4. Select All.
5. Click the Stop button.

8.4.1.2 Stopping Specific Instance of OAS

Use this technique if you wish to stop only some OAS HTTP Listeners and leave others running:

1. Launch OAS.
2. Open the OAS Manager.
3. Navigate to HTTP Listeners.
4. Select those HTTP Listeners running on ports you are planning to convert from cartridge to CGI.
5. Click the Stop button.

8.4.2 Configuring the formsweb.cfg File

The formsweb.cfg file is a powerful new convenience included with Oracle Forms Server 6*i*. Use it as a repository for all the settings you need to run Oracle forms on the Web in a CGI implementation. The installer places this file in $ORACLE_HOME/6iserver/forms60/server.

The formsweb.cfg file is a text file that contains configuration parameters for running Forms applications on the Web in a CGI implementation. The configuration parameters in the formsweb.cfg file are the CGI-equivalent of the cartridge parameters used with the Forms cartridge. The formsweb.cfg file is divided into three main sections:

- System Parameters
- User Parameters
- Specific Configurations

Refer to Chapter 5.5.1, "formsweb.cfg" for more specific information about configuring the formsweb.cfg file.

8.4.2.1 System Parameters

The System Parameters section provides information required by the Forms Web CGI. Unlike many other parameters in formsweb.cfg, System Parameters cannot be specified in a URL query string. However, you can override their values by placing an alternate parameter/value set in a Specific Configuration section in formsweb.cfg, then calling that configuration in the application URL.

8.4.2.2 User Parameters

The User Parameters section is where you specify the actual values for parameters that are defined with variables in the base HTML file. For example, in the base.htm file you might have:

<PARAM NAME="separateFrame" VALUE="%separateFrame%">

In the formsweb.cfg you would set the specific value for the variable %separateFrame%:

separateFrame=false

You can override specified User Parameter values in a Specific Configuration section in formsweb.cfg or in a query string in the application's URL.

For example:

NT:

```
http://servername.my.domain.name.com/dev60cgi/ifcgi60.exe?separateFrame=true
```

UNIX:

```
http://servername.my.domain.name.com/dev60cgi/f60cgi?separateFrame=true
```

In these examples, the query string ?separateFrame=true will override the value for separateFrame that is specified in the formsweb.cfg file.

When a specific value for a parameter is defined in both the formsweb.cfg file and the application's URL, the value defined in the URL is used.

8.4.2.3 Specific Configurations

If you want to run the same form with multiple configurations, you can define custom configurations with custom values in the Specific Configurations section of the formsweb.cfg file.

When you call the custom configuration with a query string in the application's URL, the custom values will override the parameters defined in the User Parameters section of formsweb.cfg. When you set up a Specific Configurations section, you need only specify the parameters you want to change. The default values that are specified in the User Parameters section will be used for all other parameters.

Use the "config" parameter in the application's URL to call a particular Specific Configuration section. For example, the following URLs call the Specific Configuration section [myconfig]:

NT:

```
http://servername.my.domain.name.com/dev60cgi/ifcgi60.exe?config=myconfig
```

UNIX:

```
http://servername.my.domain.name.com/dev60cgi/f60cgi?config=myconfig
```

Refer to Chapter 5.5.1, "formsweb.cfg" for more specific information about configuring the formsweb.cfg file.

8.4.3 Configuring the base.htm or basejini.htm File

When you start a Web-enabled application (by clicking a link to the application's URL), the Forms CGI reads a special file that contains all necessary applet tags, parameters, and parameter values (or variables for those values) that are required to run the selected application on the Web. This is the base HTML file.

The Oracle Universal Installer places two base HTML files in the following directory: $ORACLE_HOME/FORMS60/server

- basejini.htm

  This file contains the tags required to run the Forms applet using a combination of the user's Web browser and Oracle JInitiator.
- base.htm

  This file contains the tags required to run the Forms applet in the AppletViewer or in any Web browser certified by Oracle whose native JVM is certified to work with Forms.

Refer to Chapter 5.5.2, "base.htm and basejini.htm" for more specific information about the base.htm and basejini.htm files.

In a Forms Web CGI implementation, as the application launch process gets started, any variables (%variablename%) in the base HTML file are replaced with the appropriate parameter values that are specified either in the formsweb.cfg file or in a query string included in the application's URL. Once all values are defined, the HTML file is generated and then downloaded to the user's Web browser, and the selected forms application launches.

When a specific value for a parameter is defined in both the formsweb.cfg file and the application's URL, the value defined in the URL is used.

In most cases, you will not need to modify the default base HTML files. Instead, you can define their parameters with variables. Then you can define the actual values for the variables in formsweb.cfg or in the application's URL.

For example, you can define the parameter splashScreen in the base HTML file as:

<PARAM NAME="splashScreen" VALUE="%splashScreen%">

Then define the actual value in the formsweb.cfg file as:

NT: splashScreen=directory path\mysplashscreen.gif

UNIX: splashScreen=directory path/mysplashscreen.gif

If you prefer, you can define the splashScreen variable in the application's URL:

NT:

```
http://servername.my.domain.com/dev60cgi/ifcgi60.exe?splashScreen=directory
path/mysplashscreen.gif
```

UNIX:

```
http://servername.my.domain.com/dev60cgi/f60cgi?splashScreen=directory
path/mysplashscreen.gif
```

Using variables instead of values in the base HTML file allows you to use the same generic base HTML file for all your forms applications and to manage configuration complexity from one location: formsweb.cfg (or the application's URL).

If you decide to specify parameter values in the base HTML file, do not modify the original base HTML file that is provided by Oracle. Instead, modify a renamed copy. Be sure to update the baseHTML (or the basejiniHTML) parameter in the formsweb.cfg file to point to the location of the modified file.

8.4.4 Broadcasting the Applications's URL

To broadcast the application's URL, simply notify your intended users. Your users can contact the URL with their Java-enabled Web browsers and run the corresponding application. For example, to announce the availability of its new Order Tracking application, ABC Corp. might notify employees via e-mail of the following URL:

NT:

```
http://servername.my.domain.name.com/dev60cgi/ifcgi60.exe?config=myconfig&form=tracker.fmx
```

UNIX:

```
http://servername.my.domain.name.com/dev60cgi/f60cgi?config=myconfig&form=tracker.fmx
```

ABC's URL consists of the following components:

http	**Connection Protocol**
servername	Name of the machine that hosts the application server
my.domain.name.com	Name of the domain that hosts the target information
dev60cgi	The virtual path, defined in the Web server, that points to CGI executables (scripts)
ifcgi60.exe (NT) f60cgi (UNIX)	Forms CGI, which is used for load balancing
?config=myconfig&form=tracker.fmx	The query string that points to a custom configuration defined in the user-created "myconfig" section of the formsweb.cfg file and to the form module tracker.fmx The parameter "form" is used here because "form" was defined as the variable value for "module" in the base HTML file. That is: <PARAM NAME="serverArgs" VALUE="module=%form%"> The syntax is slightly different in the base HTML JInitiator file: serverArgs="module=%form%"

8.5 Guidelines for Migration

When migrating your applications from client/server deployment to the Web, note that a Web-based application:

- Supports JPEG and GIF image types only, so convert existing images to these formats.
- Supports the use of compressed JAR (Java Archive) files for file transfer, so use JAR files whenever the transfer of large files is required between the Forms Server and Java client.
- Does not support ActiveX, OCX, OLE, or VBX controls in the user interface. Instead, use JavaBeans to duplicate functionality in the user interface. Any other Microsoft Windows user interface dependencies should also be replaced with JavaBeans.
- Does not support MouseMove triggers, such as When-Mouse-Enter, When-Mouse-Leave, and When-Mouse-Move.
- Does not natively support write access to the client hard drive. This can be accomplished by writing a JavaBean for the pluggable Forms user interface.
- Supports Java fonts only, so check applications for the types of fonts used. If necessary, switch to Java fonts. Java uses a font alias list, located in the Registry.dat file. The font aliases described in Table 8–1 are supported:

Table 8–1 Font support for Web-based applications

Java font	Windows font	XWindows font	Macintosh font
Courier	Courier New	adobe-courier	Courier
Dialog	MS San Serif	b&h-lucida	Geneva
DialogInput	MS San Serif	b&h-lucidatypewriter	Geneva
Helvetica	Arial	adobe-helvetica	Helvetica
Symbol	Wingdings	itc-zapfdingbats	Symbol
Times Roman	Times New Roman	adobe-times	Times Roman

In this chapter we have provided information about reconfiguring forms cartridge implementations to CGI. We've kept to a fairly narrow path of configuration options to ensure a smooth and successful migration.

9

Network Considerations

9.1 Introduction

For the best implementation of Forms Server, you need to determine:

- The type of network on which you will deploy Web applications
- How your network and security issues will be managed
- The number and types of users that you expect will need to access your network

This chapter describes the types of networking implementations upon which you can deploy Web applications, and the things you need to consider when deploying Web applications on each type.

9.2 Network Topologies

There are a number of terms used to describe the various networking implementations upon which you can deploy applications. In general, networks can be grouped into the following categories:

- *Internet* is a network that is open to anyone with access to an Internet Service Provider (ISP). It uses data transmission standards drafted by the Internet Engineering Task Force (IETF).
- *Intranet* is a network that is "owned" by a single organization that controls its security policies and network management.
- *Extranet* is a network that is "owned" by multiple organizations, each of which may have their own network infrastructure, security policies, and users, thereby requiring an integrated approach to network management and security.

The primary difference between the Internet, intranets, and extranets is that an intranet and extranet are well defined by the controlling organization(s) and have a known body of users. Conversely, the Internet has an unknown body of users. Computers and networks that communicate via the Internet are unknown to each other until the time of connection. This means that there can be no previous coordination of encryption standards, user authentication, authorization, and so on.

These implementations are discussed in greater detail in the following sections:

- Internet
- Intranet
- Extranet

9.2.1 Internet

The *Internet* is a network that is open to anyone with access to an Internet Service Provider (ISP). By connecting to the Internet, a user has access to other networked computers all over the world. If a computer that is connected to the Internet is not secured using hardware or software security methods, data on that computer is potentially accessible to anyone on the Internet.

9.2.2 Intranet

An *intranet* is a network that is "owned" by a single organization that controls its security policies and network management. Networked computers may be housed within a single physical location (for example, computers used for inventory control in a manufacturing plant), or they may be in different physical locations (for example, computers used at various branches of an insurance company).

Because the intranet is controlled by a single organization, all users who will attempt to access the network are known, and there is freedom in selecting the network structure, security policy, and software.

The following are examples of intranet-style networks:

- Local-area network (LAN)
- Wide-area network (WAN) that is comprised of a LAN that extends usage to remote employees with dial-up access
- WAN that is comprised of interconnected LANs using dedicated communication lines

- Virtual private network (VPN) that is comprised of a LAN or WAN that extends usage to remote employees or networks using special "tunneling" software that creates a secure, usually encrypted connection over public lines, sometimes via an Internet Service Provider (ISP)

9.2.3 Extranet

An *extranet* is a network that is "owned" by multiple organizations, each of which may have their own network infrastructure, security policies, and users. The networked computers are usually housed in different physical locations. In most cases, the different organizations share portions of their network data with each other. For example, the travel industry uses an extranet that allows travel agents to book flights and make other travel arrangements using data from networks owned by airlines and tour operators.

Like an intranet, there is a known body of users in an extranet. However, because the extranet is controlled by multiple organizations, an integrated approach to network management and security is required. In the travel industry example, the travel agencies and airlines would have to coordinate networking and security issues in order for travel agents to access airline booking information.

The following are examples extranet-style networks:

- LANs or WANs belonging to multiple organizations and interconnected and accessed using remote dial-up
- LANs or WANs belonging to multiple organizations and interconnected and accessed using dedicated lines
- Virtual private network (VPN) that is comprised of LANs or WANs belonging to multiple organizations, and that extends usage to remote users using special "tunneling" software that creates a secure, usually encrypted network connection over public lines, sometimes via an ISP

Organizations sharing networked data and applications via an extranet must agree on the security protocols for user authentication, authorization, and data encryption. Security hardware, such as firewalls and routers, must be compatible.

9.3 Deploying Forms Server in your Network Environment

After studying how the Forms Server functions and determining the type of network setup that would work best for your company, you can implement Forms Server on your network. The following five sections describe networking options and some associated risks:

- Deploying Over the Internet
- Deploying On a Local Area Network (LAN)
- Deploying On a Network with Remote Dial-Up Access
- Deploying On a Network via Telecom-Provided VPN Access over Public Lines
- Deploying On a Network via VPN Access over the Internet

9.3.1 Deploying Over the Internet

Forms Server allows you to deploy your Forms applications over the Internet by encapsulating Forms messages in HTTP 1.1 packets. HTTP is one of the most widely used protocols for deploying applications on the Internet.

Many organizations have "locked-down" their firewalls by allowing only HTTP traffic, which greatly enhances the security of their private networks. (Most firewall companies support the HTTP standard in their products, and many organizations are willing to allow HTTP traffic in and out of their private networks.) Sites that allow only HTTP traffic will be able to easily deploy Forms Server through their existing firewall with little or no change to their configuration and with complete transparency to the client.

Although a strict security policy is still required to protect the internal company network, you can put application servers behind a firewall and in a demilitarized zone (DMZ) within the company network. The HTTP filter within the firewall is sufficient to restrict incoming traffic without the use of a VPN.

In addition, you can use SSL (secure sockets layer) with HTTP for even more secure communications. SSL is a transport protocol that provides privacy, integrity, and authentication. SSL works at the transport level, which is one level below the application level. This means that SSL can encrypt and decrypt messages before they are handled by application-level protocols such as HTTP.

Deploying Forms Server on the Internet makes your application available to individual users on the Web, as well as to extranet customers, at a relatively low cost when compared to the other network deployment options. It enables organizations to run scalable, secure, and sophisticated new or existing Forms applications over the Internet.

9.3.1.1 Risks

To deploy applications on the internet with an HTTP socket connecion, CPU requirements for the user’s Forms Client PC are slightly higher than for previous versions of Forms Server in order to provide equivalent performance.

Sending Forms data in an HTTP wrapper will likely increase network traffic, and may have an impact on the number of sessions that can be run simultaneously on lower speed connections.

9.3.1.2 Other Internet Deployment Options

If you do not choose to use the HTTP socket connection method, your other option is to set up a DMZ outside of your protected network that contains the application server. You can set up an IP-router to block all incoming packets except those destined for ports 80 (HTTP traffic) and 9001 (default port for the Forms Listener) in order to protect the DMZ. The risk with this approach is that the Forms Server Listener port is still vulnerable. If multiple Forms Server Listeners are used (for example, when hosting multiple applications or multiple languages) the risks increase.

In addition, the IP router should be backed by a multi-homed firewall residing in the DMZ that re-routes all incoming traffic from the IP router to the application servers in the DMZ. The application servers need to connect to the database in the trusted corporate network, so the multi-homed firewall also needs to re-route all Net8 traffic to the data server in the trusted corporate network.

A rotation schedule can be set up where different Forms Server Listeners are used at different times to reduce the chance of break-in, although this will not deter a serious hacker.

To shield the internal network from attacks, we recommend that you set up an extra firewall between the multi-homed firewall and the internal network to filter the IP packets and only pass Net8 traffic.

9.3.2 Deploying On a Local Area Network (LAN)

If all users who will access your Forms applications are located within your LAN, then basic internal network security is sufficient, and the Forms Server will not require any special configuration.

9.3.3 Deploying On a Network with Remote Dial-Up Access

If some users are located outside your LAN or secure WAN and will dial in for access to your Forms applications, then you will need a server designed specifically for remote access security. This scenario is ideal for employees who work offsite or for trusted customers who must access your LAN or WAN. This solution is not appropriate for implementations where more than 1000 users would need to access the LAN remotely.

Valid users are those who have been registered in your remote access server. Unregistered users do not have access. Remote Access Service (RAS) is a feature of Windows NT servers. A Windows NT RAS server can be used in this scenario as the remote access server.

A private WAN is often constructed with leased lines.To break in, an intruder would have to know the location of the leased lines and the wire codes of the lines used to transmit data. Under these conditions, a breach is unlikely.

If dial-up is via public phone lines, we recommend that you encrypt confidential data during transmission. Windows NT RAS servers include the Point-to-Point-Tunneling Protocol (PPTP), which can be used for encryption of confidential data over public dial-up lines. If you are not using a remote access server that provides an encryption protocol, see the following sections for other, more secure options for configuring Forms Server on your network.

There is a very small risk that an intruder can randomly dial the phone number for a remote access server, and then attempt multiple username/password combinations to log in to the LAN. However, remote access servers are more vulnerable to disgruntled ex-employees or customers who already know how to access the server.

To avoid this situation, we recommend the following precautions:

- Rigorous security record maintenance, which will ensure that entries for former employees and customers are removed from the remote access server, auto-dialback unit, and all internal systems
- Caller ID verification, which is a technique that only allows registered phone numbers to reach the remote access server
- Auto dial back unit, which calls back the caller using a previously registered phone number

9.3.4 Deploying On a Network via Telecom-Provided VPN Access over Public Lines

As mentioned in the previous section, a conventional WAN is usually constructed with leased lines. However, if dial-up is via public phone lines, we recommend that you have a more secure method of user authentication and data transmission.

One option is to use a VPN, or virtual private network, available from your telecommunications provider. The telecommunications provider keeps a list of allowed users, and creates the VPN whenever an approved user dials in. Your network would still need a remote access server, as described in the previous section, so all of the security benefits and risks of the previous section apply here.

(This solution is not appropriate for implementations where more than 1000 users would need to access the LAN remotely.)

The primary risk is vulnerability to disgruntled ex-employees or customers who already know how to access the server and are already on the VPN provider's registered users list. To eliminate this risk, be sure to keep current the list of approved users for both the remote access server and the VPN provider's registered users list.

9.3.5 Deploying On a Network via VPN Access over the Internet

If you plan to use the Internet as your means of dial-up access, we recommend that you have a secure method of user authentication and data transmission. One option is to use the Forms Server HTTP socket configuration, or HTTPS (HTTP socket configuration with secure sockets layer for improved privacy, integrity, and authentication.) For more information about HTTP sockets, see Section 3.2, "Sockets, HTTP, or HTTPS".

Another option is to use a VPN over the Internet. With this method, data is transferred over the Internet in the form of IP (Internet protocol) packets. An IP packet is a group of bits (your data) along with a source and destination IP address.

If you set up a VPN over the Internet, you can save telecommunication costs. Remote users dial a local ISP rather than leased lines or an 800 number. You must configure and maintain the VPN software at your network, and the users who dial in must have compatible VPN software. If you set up an extranet connection where two LANs communicate via the Internet, all parties need to use compatible firewalls. If you have remote workers, some vendors offer mobile firewalls that can be used by remote workers; however, this adds significant cost and administrative time.

Most major firewall vendors have options for implementing a VPN over the Internet. Preferred VPNs use:

- Strong user authentication, which includes a challenge / response mechanism rather than simply a username / password mechanism
- Internal firewalls to control the access to more secure parts of the network
- Data encryption to protect the data during its transport across the public network (This is called "IP tunneling," where the data in each IP packet is encrypted during its transport across the public network and decrypted at the destination.)

Risks involved with setting up a VPN over the Internet include:

- If you do not use an HTTP socket connection, then your firewall may not allow data to pass. In some cases, you can configure your firewall and Forms Server to work around this problem by setting up a generic proxy.
- Network performance is likely to degrade because of the extra processing required for strong authentication and data encryption.
- Keys must be properly configured and managed.
- Firewall configuration must be strictly managed so that ex-employees and ex-customers are de-registered.
- Spoofing the firewall is a potential risk. (Spoofing is when an intruder arrives disguised as a trusted node on the network by forging a false address in IP packets, and sending those packets to your network. The intruder gets the false address by monitoring the traffic on your network and determining addresses that have been accepted by your network.) You can deter spoofing by using filters on your firewall.

9.4 Guidelines for Maintaining Network Security

If you are planning to implement a mission-critical application using Forms Server, security is a key issue. After determining the type of network environment you need, formulate a security policy to protect it. Refer to Chapter 10, "Security Considerations" for more detailed information.

After your application servers are up and running, you must continually maintain security. This is true particularly if your applications are accessed through the Internet because your site will likely be visited by hackers. The enforcement of a security policy is an ongoing process.

We have described several deployment options for intranet, extranet, and Internet Forms applications, and have looked at the associated impact on security. From this we can draw the following conclusions:

- Intranet and extranet implementations using a dial-up WAN or dial-up VPN can be made reasonably safe with medium effort. As with a LAN, most attacks will be from the inside, so it pays to improve server protection and database user management. Encryption mechanisms should be used to protect confidential data from unauthorized users.
- For intranet and extranet implementations over an Internet VPN, use strong authentication and encryption, as well as strong access control. Most major

firewall vendors have VPN options to block access to unauthorized users, encrypt data over public networks, and provide user authentication.

A realistic implementation of security measures on the Internet is based on a combination of the following elements:

- HTTP or HTTPS socket communications
- Application servers in a DMZ
- Firewalls that shield the internal network from the DMZ
- Data encryption wherever possible

10

Security Considerations

10.1 Introduction

Before the great explosion of interest in the World Wide Web, it was common practice to run utilities or programs on the Internet that would interrogate specified remote computers to locate friends or colleagues and see if they were logged on. You could then communicate with them in real-time over the network or connect temporarily to their disk drive to exchange files.

The Internet was virtually wide open, operating with a high level of trust and a low level of security. Now, because there are millions of users, security has become a huge concern. Companies are securing their networks to prevent uncontrolled or unsolicited access to their private networks from the outside.

This chapter explores some of the issues surrounding network security.

10.2 Common System Security Issues

The following sections discuss common security issues that you must consider when setting up Forms Server in a networked environment:

- User Authentication
- Server Authentication
- Authorization
- Secure Transmission (Encryption)
- Firewall
- Virtual Private Network (VPN)
- Demilitarized Zone (DMZ)

10.2.1 User Authentication

Authentication is the process of verifying that a user who logs into a network or database has permission to log in. Examples of authentication include the use of a user name and password when logging into a local-area network (LAN) and the use of digital certificates when sending or receiving secure e-mail over the Internet. An organization can use various types of authentication processes depending on the level of security desired and the type of network or database that is being protected. But in the end, the goal of authentication is to ensure that only approved users can access the network or database and its resources.

In the case of Forms Server, running a Forms application over the Web resembles the traditional client/server environment, where the application user logs on as a database user by identifying him- or herself using a username/password combination.

Because Forms Server allows you to deploy your Forms applications to hundreds of users over the Internet, there is a risk that unauthorized users may illegitimately capture data being transmitted on a network (via a *sniffer*), intercept authentication information, and gain access to applications or the server environment. Therefore, you must implement additional security features, such as encryption and firewalls, when deploying applications over the Internet.

10.2.2 Server Authentication

With server authentication, a client machine verifies that a server is who it claims to be. For example, when a client sends confidential data to a server, the client can verify that the server is secure and is the correct recipient of the client's confidential data.

If you use the HTTPS communications mode, which uses HTTP with SSL (secure sockets layer), data transmission is encrypted and server authentication is conducted. Server authentication is accomplished using RSA-compliant digital certificates. When a client browser connects to a server, the server presents its certificate. Clients and servers get certificates from certifying authorities (CAs). CAs are companies that issue certificates to individuals or companies only after verifying the individual or company's identity. An example of a CA is VeriSign, Inc. If you decide to use HTTPS mode, you will need to install Oracle Wallet Manager in order to create certificate requests and manage certificates. See Section 5.6, "Additional Steps to Set Up the HTTPS Connection Mode" for details.

10.2.3 Authorization

Authorization is the process of giving authenticated users access to the network or database resources they need. It also prevents them from accessing resources they don't need or don't have permission to use. For example, a manager may be authorized to access tables that contain employee payroll information, but a stock clerk would not be authorized to access this information. The methods used to enforce network and database resource authorization vary depending on the level of security desired and the type of network or database being protected.

In the case of Forms Server, when a user is authenticated, a database role is assigned to the user, which grants permission to view or modify data in the database. (This is a form of authorization.) The user's identity is also used to set application roles.

10.2.4 Secure Transmission (Encryption)

When information is transmitted over lines of communication, whether they be coaxial cable, telephone lines, fiber optics, or satellite, there is the risk that the communication can be intercepted by third parties. Often, the information can be intercepted without the sender or receiver ever knowing the data was compromised.

The most common method of securing transmission is to encrypt the data. When encryption is used, the sender and receiver of the data have a "key" that can encode and decode the information. When the data is sent, the sender's key is used to encode the information using a mathematical algorithm. The receiver's key decodes the information. If a third party intercepts the encoded data while it is in transit, the data is illegible and useless unless the third party gains access to the key or "cracks" the algorithm's code.

The methods used to encrypt data vary depending on the level of security desired and the type of network over which the data is being transmitted. For example, symmetric encryption can be used if network speed is paramount. Popular symmetric cryptosystems use RC-4 and Data Encryption Standard (DES). Asymmetric encryption is highly secure, but costs in network performance. Popular asymmetric cryptosystems use Diffie-Hellman (DH) and Rivest Shamir Adlemen (RSA).

You should research the encryption methods included with your network, firewall, and/or VPN. Forms Server provides the following encryption options to improve data transmission security:

- HTTPS communication mode: This mode which HTTP with SSL (secure sockets layer). SSL encrypts and decrypts messages (using RC4 encryption) before they

are handled by application-level protocols such as HTTP. It also provides RSA-compliant server authentication. See Section 5.6, "Additional Steps to Set Up the HTTPS Connection Mode" for information about how to set up HTTPS mode.

- ORA_ENCRYPT_LOGIN: Use this environment variable to encrypt usernames and passwords for Forms Server login.
- DBLINK_ENCRYPT_LOGIN: Use this environment variable to encrypt usernames and passwords for database login.
- FORMS60_MESSAGE_ENCRYPTION: Use this environment variable to encrypt Forms messages using RC4 40-bit encryption. Applies only to socket and HTTP communication modes. (By default, communication is encrypted.)
- FORMS60_HTTPS_NEGOTIATE DOWN: Use this environment variable to direct 128-bit servers on how to handle clients that are configured for lower-level encryption. A TRUE setting will cause the server to use the highest level of encryption available to the client. A FALSE setting will cause the server to reject the client requests unless the client uses 128-bit encryption.
- DSA (Digital Signature Algorithm): This algorithm is used by the Forms Server applet for digital signatures.
- Net8 SNS/ANO: This encryption scheme is used to encrypt transmission between the database and Forms Server.

10.2.5 Firewall

A firewall is usually a combination of hardware and software that filters the types of data that can be received by your network. For example, a firewall can be configured to allow only HTTP traffic through to the protected network. A firewall also keeps your network's IP address anonymous so that it is not accessible to outside computers. Outside traffic that is authenticated and permitted access to your network is redirected from the firewall IP address to the network IP address. The firewall is your private network's first line of defense against intrusion.

If your network security system includes a firewall, be sure to configure the Forms Server listener to use the HTTP socket connection or HTTPS socket connection rather than the standard socket connection. This is because a firewall will disable many common services at the packet or port level, including standard Forms messaging. HTTP is a service that is allowed to pass through firewalls.

10.2.6 Virtual Private Network (VPN)

A Virtual Private Network (VPN) is an authenticated connection between two networks or between a network and a remote user where communication is considered completely private. Special "tunneling" software on both the network and the remote user's computer create a secure, encrypted connection over public lines — even via an Internet Service Provider (ISP). If the remote user does not have the appropriately configured VPN software, it cannot create a VPN with the network.

Often, a VPN setup includes a firewall. Be sure to configure the Forms Server listener to use the HTTP socket connection or HTTPS socket connection rather than the standard socket connection. This is because a firewall will disable many common services at the packet or port level, including standard Forms messaging.

Note: For more information on HTTP and sockets, see Chapter 3.2, "Sockets, HTTP, or HTTPS".

10.2.7 Demilitarized Zone (DMZ)

A Demilitarized Zone (DMZ) is an isolated environment in your network that does not contain confidential information. For example, you may have a network where application servers are within the demilitarized zone, but all database servers are within the protected network. Then, if the demilitarized zone's security is compromised, confidential data is not exposed to the intruder.

10.3 Simple Steps to Improve Security

Here are some steps that can help reduce the risks associated with network security:

- Discourage users from lending their username / password to unauthorized users.
- Enforce a strict authorization scheme with clear database roles that match various user profiles, such as Order Entry Clerk, Executive Officer, Product Marketer, and so on. Each role restricts permissions to modify or even view data according to the user profile.
- Carefully manage user accounts by removing users who no longer need to access servers or databases and by enforcing password aging.
- Use the HTTPS connection mode for encryption and digital certificate authentication.
- Use ORA_ENCRYPT_LOGIN and DBLINK_ENCRYPT_LOGIN to encrypt the usernames and passwords that are being transmitted.

- Use encryption, such as FORMS60_MESSAGE_ENCRYPTION and Net8 SNS/ANO, whenever possible to avoid exposing confidential data to intruders.

The following are network security considerations that seem obvious, but are often overlooked:

- Control physical access to server machines so that unauthorized people cannot enter the building and access them.
- Implement a rigorous data backup system, including the secure storage of backup media.
- Remove or minimize the use of easily compromised services such as telnet and ftp.
- Install all security-related operating system patches.

11

Performance Tuning Considerations

11.1 Introduction

This chapter describes the tuning considerations that arise when you use Forms Server to deploy an application over the Internet or other network environment. This chapter looks at the network and the resources on the application server. Tuning the connection between Forms Server and the database server is beyond the scope of this chapter.

11.2 Built-in Optimization Features of Forms Server

The Forms Server and Java client include several optimizations that fit broadly into the following categories:

- Minimizing Client Resource Requirements
- Minimizing Forms Server Resource Requirements
- Minimizing Network Usage
- Maximizing the Efficiency of Packets Sent Over the Network
- Rendering Application Displays Efficiently on the Client

11.2.1 Minimizing Client Resource Requirements

The Java client is primarily responsible for rendering the application display. It has no embedded application logic. Once loaded, a Java client can display multiple Forms simultaneously. Using a generic Java client for all Forms Server applications requires fewer resources on the client when compared to having a customized Java client for each application.

The Java client is structured around many Java classes. These are grouped into functional subcomponents, such as displaying the splash screen, communicating with the network, and changing the look-and-feel. Functional subcomponents allow the Forms Developer and the Java Virtual Machine (JVM) to load functionality as it is needed, rather than downloading all of the functionality classes at once.

11.2.2 Minimizing Forms Server Resource Requirements

When a Form definition is loaded from an FMX file, the profile of the executing process can be summarized as:

- Encoded Program Units
- Boilerplate Objects / Images
- Data Segments

Of these, only the Data Segments section is unique to a given instance of an application. The Encoded Program Units and Boilerplate Objects / Images are common to all application users. Forms Server maps the shared components into physical memory, and then shares them between all processes accessing the same FMX file.

The first user to load a given FMX file will use the full memory requirement for that Form. However, subsequent users will have a greatly reduced memory requirement, which is dependent only on the extent of local data. This method of mapping shared components reduces the average memory required per user for a given application.

11.2.3 Minimizing Network Usage

Bandwidth is a valuable resource, and the general growth of Internet computing puts an ever increasing strain on the infrastructure. Therefore, it is critical that applications use the network's capacity sparingly.

Forms Server communicates with the Java client using meta data messages. Meta data messages are a collection of name-value pairs that tell the client which object to act upon and how. By sending only parameters to generic objects on the Java client, there is approximately 90-percent less traffic (when compared to sending new code to achieve the same effect).

Forms Server intelligently condenses the data stream in three ways:

- When sets of similar messages (collections of name-value pairs) are sent, the second and subsequent messages include only the differences from the previous message. This results in significant reductions in network traffic. This process is called *message diff-ing*.
- When the same string is to be repeated on the client display (for example, when displaying multiple rows of data with the same company name), Forms Server sends the string only once, and then references the string in subsequent messages. Passing strings by reference increases bandwidth efficiency.
- Data types are transmitted in the lowest number of bytes required for their value.

11.2.4 Maximizing the Efficiency of Packets Sent Over the Network

Latency can be the most significant factor that influences the responsiveness of an application. One of the best ways to reduce the effects of latency is to minimize the number of network packets sent during a conversation between the Java client and the Forms Server.

The extensive use of triggers within the Forms Developer model is a strength, but they can increase the effect of latency by requiring a network round trip for each trigger. One way to avoid the latency concerns adhering to triggers is by grouping them together through Event Bundling. For example, when a user navigates from item A to item B (such as when tabbing from one entry field to another), a range of pre- and post-triggers may fire, each of which requires processing on the Forms Server.

Event Bundling gathers all of the events triggered while navigating between the two objects, and delivers them as a single packet to the Forms Server for processing. When navigation involves traversing many objects (such as when a mouse click is

on a distant object), Event Bundling gathers all events from all of the objects that were traversed, and delivers the group to the Forms Server as a single network message.

11.2.5 Rendering Application Displays Efficiently on the Client

All boilerplate objects in a given Form are part of a Virtual Graphics System (VGS) tree. VGS is the graphical subcomponent that is common to all Forms Developer products. VGS tree objects are described using attributes such as coordinates, colors, line width, and font. When sending a VGS tree for an object to the Java client, the only attributes that are sent are those that differ from the defaults for the given object type.

Images are transmitted and stored as compressed JPEG images. This reduces both network overhead and client memory requirements.

Minimizing resources includes minimizing the memory overhead of the client and server processes. Optimal use of the network requires that bandwidth be kept to a minimum and that the number of packets used to communicate between the client and Forms Server be minimized in order to contain the latency effects of the network.

11.3 Tuning Forms Server Applications

An application developer can take steps to ensure that maximum benefits are gained from Forms Server's built-in architectural optimizations. The remainder of this chapter discusses key performance issues that affect many applications and how developers can improve performance by tuning applications to exploit Forms Server features.

Issues discussed are:

- Location of the Form Server with Respect to the Data Server
- Minimizing the Application Startup Time
- Reducing the Required Network Bandwidth
- Other Techniques to Improve Performance

11.3.1 Location of the Form Server with Respect to the Data Server

The Java client connection to the Forms Server can use features such as Event Bundling to effectively counteract the effects of network latency. It uses *message*

diff-ing to reduce network bandwidth. On the other hand, the client/server relationship that exists between the Forms Server and the data server is much less tolerant of round-trip delays and network congestion.

For these reasons, it is best to locate the Forms Server on the same high speed LAN as the data server, which may consequently locate the Forms Server more remotely from the users. This may seem contrary to the standard convention of placing servers in close proximity to users, but it is a consequence of Forms Server's improved efficiency over a network as compared to a traditional client/server implementation.

In an optimal configuration, as shown in Figure 11–1, the Form Server and data server are co-located in a Data Center, which is the recommended set-up, while clients access the server over low-bandwidth (modem) and high-latency (satellite) connections.

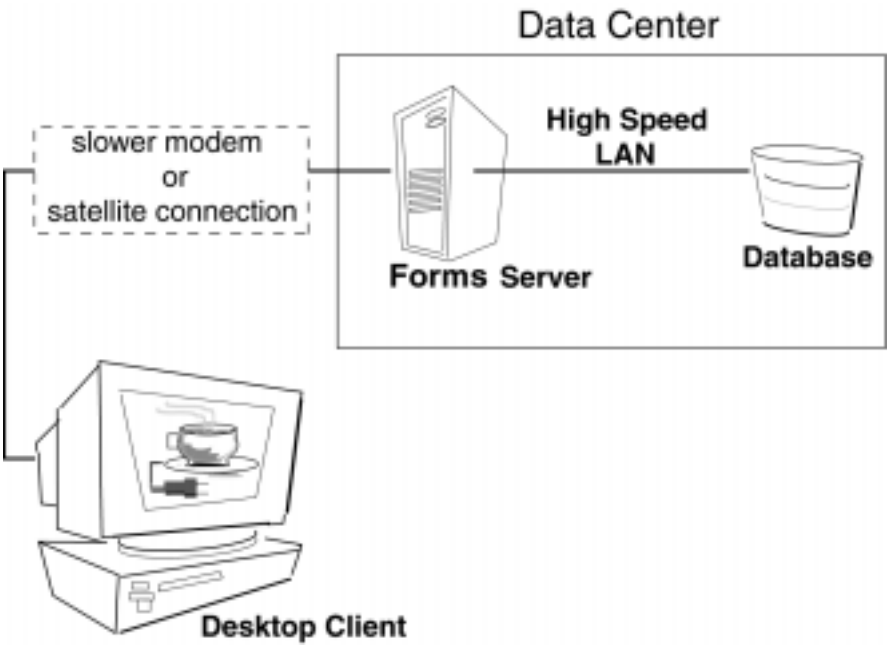


Figure 11–1 Co-Locating the Forms Server and Data Server

11.3.2 Minimizing the Application Startup Time

First impressions are important, and a key criterion for any user is the time it takes to load an application. Startup time is regarded as overhead. It also sets an expectation of future performance. When a business uses thin-client technologies, the required additional overhead of loading client code may have a negative impact on users. Therefore, it is important to minimize load time wherever possible.

After requesting a Forms application, several steps must be completed before the application is ready for use:

1. Invoke Java Virtual Machine (JVM).
2. Load all initial Java client classes, and authenticate security of classes.
3. Display splash screen.
4. Initialize Form:
    a. Load additional Java classes, as required.
    b. Authenticate security of classes.
    c. Render boilerplate objects and images.
    d. Render all elements on the initial screen.
5. Remove splash screen.
6. Form is ready for use.

An application developer has little influence on the time it takes to launch the JVM. However, the Java deployment model and the structure of the Form Developer Java client allow the developer to decide which Java classes to load and how. This, in turn, minimizes the load time required for Java classes.

The Java client requires a core set of classes for basic functionality (such as opening a window) and additional classes for specific display objects (such as LOV items). These classes must initially reside on the server, but the following techniques can be used to improve the time it takes to load these classes into the client's JVM:

- Using JAR Files
- Using Caching
- Deferred Load on Demand

11.3.2.1 Using JAR Files

Java provides the Java Archive (JAR) mechanism to create files that allow classes to be grouped together and then compressed (zipped) for efficient delivery across the network to the client. Once used on the client, the files are cached for future use.

Form Server provides the following pre-configured JAR files to support typical deployment scenarios:

File name	Usage	Description
f60all.jar	Optional	Contains the entire set of Java class files for all runtime situations.
f60common.jar	Required	Required by the applet.

f60generic_laf.jar	Optional	Must be loaded if the application is deployed with the Generic lookAndFeel runtime setting or if no lookAndFeel setting is specified.
		`<APPLET ...>`
		`<PARAM NAME="lookAndFeel" VALUE="Generic">`
		`...`
		`</APPLET>`

f60oracle_laf.jar	Optional	Must only be loaded if the application is deployed with the Oracle lookAndFeel runtime setting.
		`<APPLET ...>`
		`<PARAM NAME="lookAndFeel" VALUE="Oracle">`
		`...`
		`</APPLET>`

f60splash.jar	Required	Required by the applet.
f60tree.jar	Optional	Must only be loaded if the Forms application uses the hierarchical tree control.

To specify one or more JAR files for an applet, specify the ARCHIVE parameter in the <APPLET> tag of the referencing HTML file. For example:

```
<APPLET CODEBASE="http://www.server.com/webcode/"
ARCHIVE="f60all.jar, icons.jar"
CODE="oracle.forms..">
```

11.3.2.2 Using Caching

Both of the supported JVMs for Form Server (Oracle JInitiator and Oracle JDK) support the caching of JAR files. When the JVM references a class, it first checks the local client cache to see if the class exists in a pre-cached JAR file. If the class exists in cache, JVM checks the server to see if there is a more current version of the JAR file. If there isn't, the class is loaded from the local cache rather than from across the network.

Be sure that the cache is of proper size to maximize its effectiveness. Too small a cache size may cause valid JAR files to be overwritten, thereby requiring that another JAR file be downloaded when the application is run again. The default cache size is 20MB. This size should be compared with the size of the cache contents after successfully running the application.

JAR files are cached relative to the host from which they were loaded. This has implications in a load-balancing architecture where identical JAR files from different servers can fill the cache. By having JAR files in a central location and by having them referenced for each server in the load-balancing configuration, the developer can ensure that only one copy of each JAR file is maintained in the client's cache. A consequence of this technique is that certain classes within the JAR file must be signed to enable connections back to servers other than the one from which they were loaded. The Oracle-supplied JAR files already pre-sign the classes.

11.3.2.3 Deferred Load on Demand

One downside of the JAR method is that all classes within a JAR file need to be loaded and validated by the JVM before execution continues. A useful feature of the JAR file is the ability to refer to other JAR files, thus limiting the number of classes stored within the given archive. The JVM is able to navigate to the required JAR files in the order required by the application.

The Oracle-supplied f60splash.jar file contains enough logic to initialize the client and display a welcoming splash screen. It also contains deferred references to files that are contained in the other JAR files, which are subsequently loaded on demand. In order to use deferred load on demand, the f60splash.jar file must be the first JAR file referenced in the HTML page.

11.3.3 Reducing the Required Network Bandwidth

The developer can design the application to maximize data stream compression by using *message diff-ing*, which sends along only the information that differs from one message to another. The following steps can be taken to reduce the differences between messages:

- **Control the order in which messages are sent.** The order in which messages are sent is governed by two criteria:
  - For the initial display, the display order in the Object Navigator
  - During execution, the order of program changes to item properties

  Where the result does not impact usability, you should strive to place similar objects that are on the same canvas after each other in the Object Navigator. For example, place buttons with buttons, text items with text items, and so on. (If you use the item property Next Navigation Item, the same order of navigation will be used for the items in the Form.) By ordering similar items together on the Object Navigator, the item properties sent to the client to display the first Form will include many similar items in consecutive order, which allows the *message diff-ing* algorithm to function efficiently.

  In addition, when triggers or other logic are used to alter item properties, then you should group properties of similar items together before altering the item properties of another display type. For example:

```
set_item_property(text_item1_id, FONT_WEIGHT, FONT_BOLD);
set_item_property(text_item2_id, FONT_WEIGHT, FONT_BOLD);
set_item_property(text_item3_id, FONT_WEIGHT, FONT_BOLD);
set_item_property(button_item1_id, LABEL, 'Exit');
...
```

- **Promote similarities between objects.** Using similar objects improves *message diff-ing* effectiveness (in addition to being more visually appealing to the user). The following steps encourage consistency between objects:
  - Accept default values for properties, and change only those attributes needed for the object.
  - Use Smart Classes to describe groups of objects.
  - Lock the look-and-feel into a small number of visual attributes.

- **Reduce the use of boilerplate text.** As a developer, you should use the PROMPT item property rather than boilerplate text wherever applicable. Forms Developer 6.0 and higher includes the Associate Prompt feature, which allows boilerplate text to be re-designated as the prompt for a given item.
- **Reduce the use of boilerplate items (such as arcs, circles, and polygons).** All boilerplate items for a given Form are loaded at Form initialization. Boilerplate items take time to load and use resources on the client whether they are displayed or not. Common boilerplate items, namely rectangles and lines, are optimized. Therefore, restricting the application to these basic boilerplate items reduces network bandwidth and client resources while improving startup times.
- **Keep navigation to a minimum.** An Event Bundle is sent each time a navigation event finishes, whether the navigation extends over two objects or many more. Design Forms that do not require the user to navigate through fields when default values are being accepted. A Form should encourage the user to quickly exit once the Form is complete, which causes all additional navigation events to fire as one Event Bundle.
- **Reduce the time to draw the initial screen.** Once the Java client has loaded the required classes, it must load and initialize all of the objects to be displayed before it can display the initial screen. By keeping the number of items to a minimum, the initial screen is populated and displayed to the user more promptly. Techniques that reduce the time to draw the initial screen include:
  - Providing a login screen for the application with a restricted set of objects (such as a title, small logo, username, and password).
  - On the Form's initial display, hiding elements not immediately required. Use the canvas properties:

    `RAISE ON ENTRY = YES` (Canvas only)
    `VISIBLE = NO`

    Pay attention to TAB canvases that consist of several sheets where only one will ever be displayed. For responsive switching between tabs, all items for all sheets on the canvas are loaded, including those that are hidden behind the initial tab. Consequently, the time taken to load and initialize a TAB canvas is related to all objects on the canvas and not just to those initially visible.
- **Disable MENU_BUFFERING.** By default, MENU_BUFFERING is set to True. This means that changes to a menu are buffered for a future "synchronize" event when the altered menu is re-transmitted in full. (Most applications make either

many simultaneous changes to a menu or none at all. Therefore, sending the entire menu at once is the most efficient method of updating the menu on the client.) However, a given application may make only minimal changes to a menu. In this case, it may be more efficient to send each change as it happens. You can achieve this using the statement:

```
Set_Application_Property (MENU_BUFFERING, 'false');
```

Menu buffering applies only to the menu properties of LABEL, ICON, VISIBLE, and CHECKED. An ENABLE/DISABLE event is always sent and does not entail the retransmission of an entire menu.

11.3.4 Other Techniques to Improve Performance

The following techniques may further reduce the resources required to execute an application:

- **Restrict the use of MOUSE-UP, MOUSE-DOWN triggers.** In the Java model, an event must be triggered when a mouse button action is detected. The event is passed to the Form Server to determine whether this is a MOUSE-UP or a MOUSE-DOWN event. A given application may define only one trigger (for example, MOUSE-DOWN), but an event is still generated by the client for the associated (MOUSE-UP) event, even though there is no trigger code specified to handle the event. Mouse events are asynchronous, so they are processed outside of the usual Event Bundling model.
- **Examine timers and replace with JavaBeans.** When a timer fires, an asynchronous event is generated. There may not be other events in the queue to bundle with this event. Although a timer is only a few bytes in size, a timer firing every second generates 60 network trips a minute and almost 30,000 packets in a typical working day. Many timers are used to provide clocks or animation. Replace these components with self-contained JavaBeans that achieve the same effect without requiring the intervention of Forms Server and the network.
- **Consider localizing the validation of input items.** It is common practice to process input to an item using a When-Validate-Item trigger. The trigger itself is processed on the Forms Server. You should consider using pluggable Java components to replace the default functionality of standard client items, such as text boxes. Then, validation of items, such as date or max/min values, are contained within the item. This technique opens up opportunities for more

complex, application-specific validation like automatic formatting of input, such as telephone numbers with the format (XXX) XXX-XXXX.

- **Reduce the application to many smaller Forms, rather than one large Form.** By providing a fine-grained application, the user's navigation defines which objects are loaded and initialized from the Form Server. With large Forms, the danger is that the application is delayed while objects are initialized, many of which may never be referenced. When chaining Forms together, consider using the built-ins OPEN_FORM and NEW_FORM:
  - With OPEN_FORM, the calling Form is left open on the client and the server, so that the additional Form on both the client and the server consumes more memory. However, if the Form is already in use by another user, then the increase in server memory is limited to just the data segments. When the user returns to the initial Form, it already resides in local memory and requires no additional network traffic to redisplay.
  - With NEW_FORM, the calling Form is closed on the client and the server, and all object properties are destroyed. Consequently, it consumes less memory on the server and client. Returning to the initial Form requires that it be downloaded again to the client, which requires network resources and startup time delays. Use OPEN_FORM to display the next Form in an application unless it is unlikely that the initial form will be called again (such as a login form).

12

Load Balancing Considerations

12.1 Introduction

This chapter discusses load balancing considerations for the Forms Server. Load balancing allows you to maintain a pool of middle tier machines (a "server farm") and balance the load of server traffic among these machines. Load balancing is implemented using a CGI executable that can run on any Web server with CGI support.

This chapter contains information about the following topics:

- Load Balancing Terminology
- Load Balancing in Action
- Configuring for Forms Server Load Balancing
- Setting Up the Load Balancer Server Trace Log

12.2 Load Balancing Terminology

Here are some terms you will want to understand before you set up load balancing:

- **Forms CGI**: CGI stands for Common Gateway Interface. Forms CGI is a program that is used for load balancing. It can be used with any generic listener that supports CGI.
- **Load Balancer Server**: This is the component that keeps track of all Forms Servers in the various load balancing pools. It tracks the status of the servers in a given pool and keeps statistics indicating their loads. It is responsible for directing each Form execution request to the least loaded server that is able to service requests in the given pool.

- **Load Balancer Client**: This is the component that sends load information to the Load Balancer Server, such as the number of Forms processes that are currently running on that machine. The Load Balancer Client runs on each machine with a Forms Server.
- **Primary Node**: This is the Forms Listener (plus any related software) where all URL requests to execute Forms are addressed. If load balancing is in use, each Form execution request is routed to the least loaded machine where a Forms Server is running. It gets the least loaded machine name from the Load Balancer Server.
- **Secondary Node**: These are machines on which the Forms Server, runtime client, and load balancer client are running. Forms execution requests are directed to them from the Primary Node when load balancing is being used.

  In many cases, the Primary Node will also act as a Secondary Node (for example, if it has Forms Server installed and running on it).

12.3 Load Balancing in Action

Figure 12–1 illustrates the events that occur when you use load balancing:

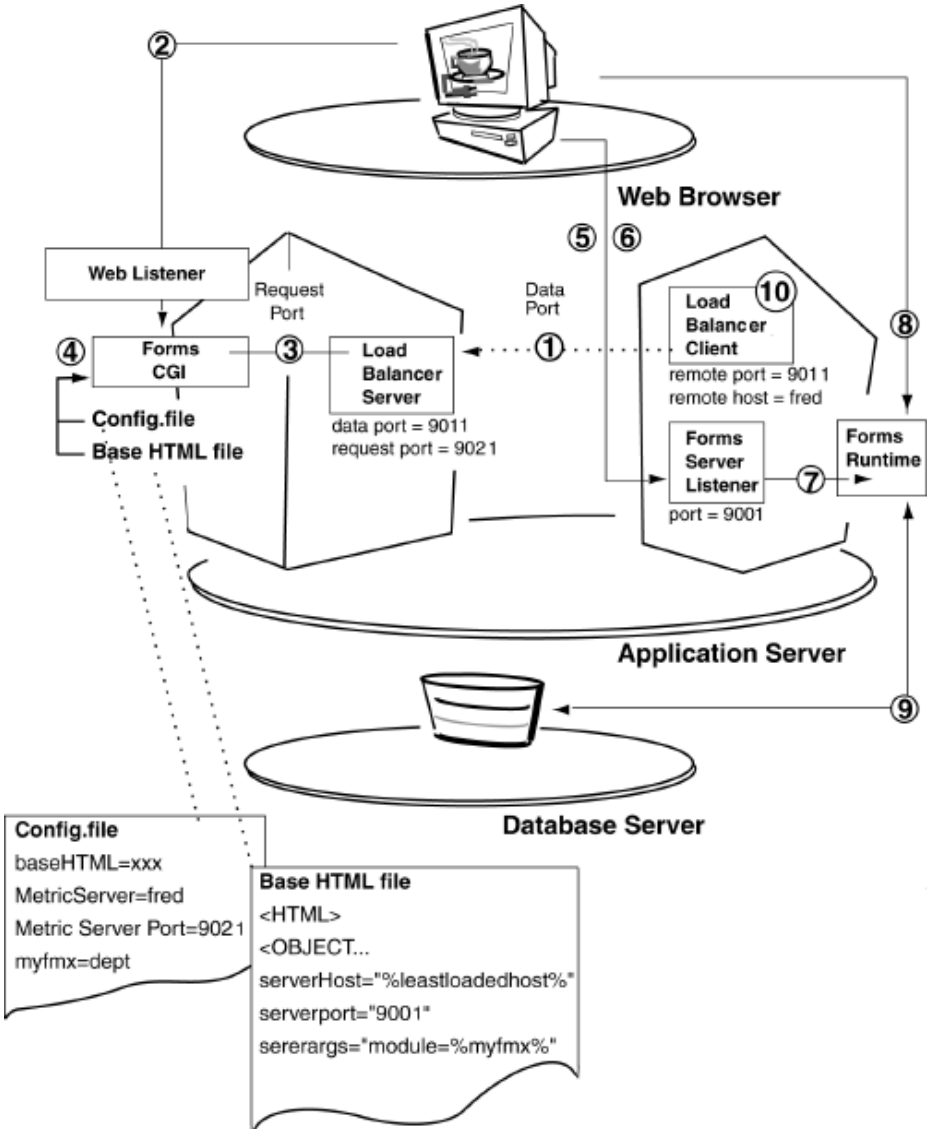


Figure 12–1 Forms Server load balancing

The following events occur when you use Forms Server load balancing:

1. Load Balancer Clients periodically send load information to the Load Balancer Server. This load information includes the total number of processes running on each Load Balancer Client.
2. A user accesses a URL pointing to the Forms CGI-bin executable.
3. The Forms CGI-bin executable asks the Load Balancer Server for the name of the least-loaded system that is available.
4. The Forms CGI-bin executable dynamically creates an HTML page with the name of the least-loaded system specified as the system on which to run the Forms Server, and returns that HTML page to the user's Web browser.
5. The user's Web browser then requests the Java applet to be downloaded from the host specified in the HTML page.
6. The Java applet sends a request to the Forms Server asking for a particular Form Builder application (that is, an .FMX).
7. The server contacts a Forms Server Runtime Engine. (The server maintains a pool of available Runtime Engines to minimize application startup delays.) Each active user receives a dedicated Runtime Engine.
8. The server establishes a direct socket, HTTP, or HTTPS connection with the Runtime Engine, and sends the socket, HTTP, or HTTPS information to the Java applet. The Java applet then establishes a direct socket, HTTP, or HTTPS connection with the Runtime Engine. The Java applet and the Runtime Engine now communicate directly, freeing the server to accept startup requests from other users. (At this point, neither the application server nor the Forms Server is involved in the communication between the applet and the Runtime Engine.) The Java applet displays the application's user interface in the main window of the user's Web browser.
9. The Runtime Engine communicates directly with the database through Net8 or ODBC (Open Database Connectivity), depending on the data source.
10. Load Balancer Clients continue to send load information to the Load Balancer Server. All new service requests are routed based on that information.

Note: If the Load Balancer Server is unavailable, at Step 3 the Forms CGI-bin executable will not get any information back about which is the least-loaded system. Instead, the Forms CGI-bin will redirect the user's browser to the URL specified by the MetricsServerErrorURL parameter. The user does not necessarily know this is happening because the redirect is behind the scenes from the user's viewpoint.

12.4 Configuring for Forms Server Load Balancing

You can implement load balancing using the following CGI-bin executables provided with Forms Server:

- Forms Server Listener (f60ctl)
- Load Balancing Server (d2ls60)
- Load Balancing Client (d2lc60)

You will need to install and configure the load balancing components on each machine that will be load balanced. This includes the machine with the primary node and any other machines containing secondary nodes.

You will also need to edit the forms60_server shell script on each machine that is using load balancing. The forms60_server shell script is found in the $ORACLE_HOME/6iserver directory.

Be sure that:

- The Data Port value for the Load Balancer Server matches the Data Port values for *ALL* Load Balancer Clients.
- All Forms Servers that are to be load balanced have the same Forms Server Port value.

You will need administrator privileges to make the changes, and will need to stop and restart the process in order for the configuration changes to take effect.

To configure for load balaning, you must set the following paramaters on each machine within the forms60_server shell script:

- Forms Server Listener Parameters
- Load Balancer Server Parameters
- Load Balancer Client Parameters

12.4.1 Forms Server Listener Parameters

Set the port number and protocol to be used by the Forms Server Listener by editing the forms60_server shell script found in the $ORACLE_HOME/6iserver directory. The following syntax is used to start the Forms Server Listener:

```
f60ctl start port=<Forms Server Port> mode=<Protocol>
```

For example:

```
f60ctl start port=9001 mode=socket
```

Forms Server Port: The default is 9001. Enter the TCP/IP port number to which the Forms Server will listen for form execution requests.

Note: All Forms Servers that are to be load balanced must have the same Forms Server Port value.

Protocol: The default is socket. This is the protocol that will be used for communication between the Forms Runtime Engine and the Forms Java applet. The value should only be changed to HTTP or HTTPS if communications must pass through a firewall. (For example, select HTTP if this machine is inside a firewall and the Forms applications must be available to users outside the firewall. Select HTTPS to use HTTP with SSL, secure sockets layer.)

You can accept the default parameters values, or modify the startup parameter values for the Forms Server. Change the port number only if it is already being used by another program.

12.4.2 Load Balancer Server Parameters

Set the port numbers to be used by the Forms Load Balancer Server by editing the forms60_server shell script found in the $ORACLE_HOME/6iserver directory. The following syntax is used to start the Load Balancer Server:

```
d2ls60 <Data Port> <Request Port> <Maximum Clients> <Trace Level>
```

For example:

```
d2ls60 9011 9021 1000 0
```

Data Port: The default is 9011. Enter the TCP/IP port number on which to listen for load data from Load Balancer Clients (which will run on Secondary Nodes).

The Data Port value for the Load Balancer Server must match the Data Port values for *ALL* Load Balancer Clients.

Request Port: The default is 9021. Enter the TCP/IP port number on which to listen for requests for the "least loaded host" made by the Forms Web CGI. This value is written to the formsweb.cfg file as the MetricServerPort parameter.

The serverHost parameter is set to the value %LeastLoadedHost% (i.e. serverHost=%LeastLoadedHost%). You should append your domain name to the serverHost parameter if a domain name is required in your network for name resolution. For example, serverHost=%LeastLoadedHost%.us.oracle.com.

Maximum Clients. The default is 1000. Specifies the maximum number of Load Balancer Clients that will be running and sending load information to the Load Balancer Server.

Trace Level: The default is 0 for no tracing. Specifying 10 allows you to create output for the Load Balancer Server.

Note: For all machines that are being used and configured as secondary nodes only, you will have to edit the forms60_server shell script and remove the section related to the Load Balancer Server.

This section includes the lines from

```
# Stop load lalancing server
```

until the line

```
# Stop load balancing client.
```

12.4.3 Load Balancer Client Parameters

Set the Load Balancer host name and data port number to be used by the Forms Load Balancer Client by editing the forms60_server shell script found in the $ORACLE_HOME/6iserver directory. The following syntax is used to start the Load Balancer Client:

```
d2lc60 <Load Balancer Host> <Data Port> 0 [<Scale Factor> <Process Name>]
```

For example:

```
d2lc60 neko 9011 0 1 f60webm
```

Load Balancer Host: The default value is originally set to the name of the local machine you installed the software on. This name needs to be changed to the name of the host containing the Load Balancer Server. Enter the full host name of the Primary Node (the machine on which the Load Balancer Server is running). The value can contain up to 256 characters.

Data Port: The default is 9011. Enter the TCP/IP port number to which the load balancer server is listening for load data. The Data Port value for each Load Balancer Client must match the Data Port value for the Load Balancer Server.

Scale Factor: The default is 4 for Windows NT and 1 for UNIX. The scale factor allows you to reduce the imbalances resulting from varying capacities of Load Balancer processes running on each Load Balancer Client. A system that appears to be the least-loaded system may not necessarily be the best place to run a new process. You should assign a higher value for the scale factor for your lower-capacity systems.

Process Name: The default is f60webm. Setting the value tells the Load Balancer Client to count processes (for load balancing pruposes_ whose executable name matches the name specified. If a value is not specified, all proceses on the machine are counted.

12.5 Setting Up the Load Balancer Server Trace Log

This section describes the format of Load Balancer Server trace messages. To start a trace, you must restart the Load Balancer Server and specify the <traceLevel> parameter in the forms60_server shell script. The <traceLevel> defaults to 0 for no tracing. Specifying 10 allows you to create trace output for the Laod Balance Server.

12.5.1 Trace level 1

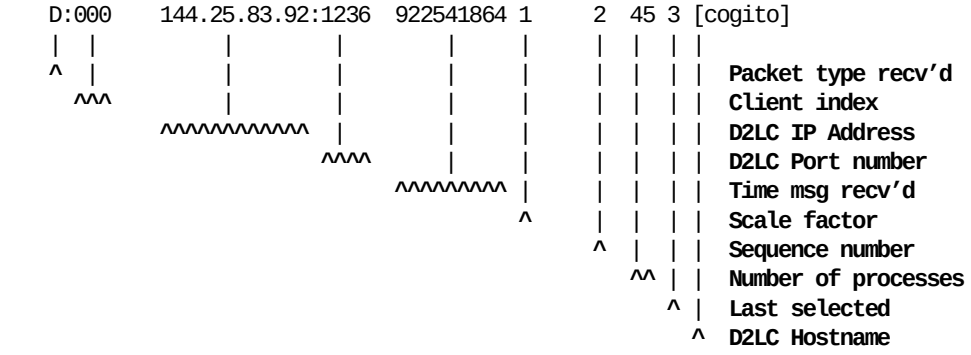
Trace level 1 contains a header as follows:

```
HOSTNAME:          neko.us.oracle.com    IP ADDRESS: 144.25.83.146
Data port number:  1234  Request port number:  1235
Maximum number of clients: 10    Trace level: 2
```

- **Hostname** and **IP Address**: Indicate the D2LS server host name and address.
- **Data port number**: Indicates the port number where the D2LS server listens for D2LC client messages. This port should be used to configure the D2LC client processes.
- **Request port number**: Indicates the port number where the server listens for requests for least loaded host information.
- **Maximum number of clients**: Indicates the number of slots allocated for D2LC clients. One slot is required for each client.
- **Trace level**: Indicates the amount of Trace information printed to the server log file.

12.5.2 Trace level 2

At Trace level 2, messages are given in the following format from a D2LC client. A description of each field follows.



The description of each field of the trace log is as follows:

- **Packet Type Received**: Indicates what type of packet was received. The following types are possible:
  - **D**: Data Received from D2LC Client. For packets with type "D", the remaining data on the trace line corresponds to information communicated from the client.
  - **S**: Selected client for Least Loaded Host. For packets with type "S", the remaining data on the trace line corresponds to the client selected and returned as the least loaded host.
- **Client index**: The client index is the internal index for the D2LC client. This index is assigned when a request is first received from the client, starting at 0.
- **D2LC IP Address**: The IP Address for the client sending the message.
- **D2LC Port number**: The IP Port number used by the client to send the message.
- **Time message received**: This is the time that the message was received from the client, in seconds since 00:00:00 UTC, January 1, 1970.
- **Scale factor**: The scale factor assigned for the client. The scale factor is used as a multiplier against the number of processes in choosing the least loaded host.
- **Sequence number**: The sequence number is the number of times the client has attempted to send messages to the D2LS server.
- **Number of processes**: The number of processes reported by the client.

- **Last selected**: This number indicates the last time a client was selected as a least loaded host. An internal counter in the server is incremented over time. When a client is selected as least loaded host, this counter is stored in the **Last Selected** field. The D2LC client with the lowest **Last Selected** field is known to be least recently used. When a request for a least loaded host results in a tie for least number of processes, then the least recently used client is selected to break the tie.
- **D2LC Hostname**: Shows the hostname of the D2LC client.

12.5.3 Sample Trace File

The following is a sample trace file for a two server configuration. Formsvr1 runs a D2L client and D2L server. Formsvr2 runs a D2L client.

```
HOSTNAME:          formsvr1.us.oracle.com       IP ADDRESS:   144.25.87.101
Data port number:  1234 Request port number:  1235
Maximum number of clients: 10   Trace level: 2

D:000   144.25.87.101:1000  925260387 1     2    0 0 [formsvr1]
D:000   144.25.87.101:1000  925260387 1     3   43 0 [formsvr1]
D:001   144.25.87.102:1001  925260388 1     2    0 0 [formsvr2]
D:001   144.25.87.102:1001  925260388 1     3   43 0 [formsvr2]
S:000   144.25.87.101:1000  925260387 1     3   44 1 [formsvr1]
D:000   144.25.87.101:1000  925260387 1     4   45 1 [formsvr1]
D:001   144.25.87.102:1001  925260388 1     4   45 0 [formsvr2]
S:001   144.25.87.102:1001  925260388 1     4   46 2 [formsvr2]
D:000   144.25.87.101:1000  925260387 1     5   45 1 [formsvr1]
D:001   144.25.87.102:1001  925260388 1     5   45 2 [formsvr2]
S:000   144.25.87.101:1000  925260387 1     5   46 3 [formsvr1]
D:000   144.25.87.101:1000  925260387 1     6   47 3 [formsvr1]
D:001   144.25.87.102:1001  925260388 1     6   47 2 [formsvr2]
S:001   144.25.87.102:1001  925260388 1     6   48 4 [formsvr2]
D:000   144.25.87.101:1000  925260387 1     7   47 3 [formsvr1]
D:001   144.25.87.102:1001  925260388 1     7   47 4 [formsvr2]
```

13

Oracle Enterprise Manager Forms Support

13.1 Introduction

This chapter describes how to install and configure Oracle Enterprise Manager (OEM) for use with Forms. It also describes the features and functions of OEM. OEM is a system management tool that consist of a graphical Java console, management server, agents, and tools that provide you with an integrated systems management platform for managing Oracle products.

This chapter contains the following sections:

- Why Should I Use OEM?
- OEM Components
- Installing and Configuring OEM Components for Use with Forms
- Managing Forms Servers from the OEM Console
- OEM Menu Options

Detailed OEM documentation is located in:

- Oracle Enterprise Manager - Concepts Guide
- Oracle Enterprise Manager - Administration Guide
- Oracle Enterprise Manager - Configuration Guide

13.2 Why Should I Use OEM?

The OEM Forms administrator interface provides the following basic functions:

- **Automatic node and service discovery**: Forms Listener, Forms Server, Load Balancer Server, and Load Balancer Client are automatically discovered by OEM's Intelligent Agent on the node to be administered, and appears in the Navigator tree of the OEM console.
- **Node and service control**: Some basic controls such as startup and shut down are provided for discovered nodes and services.
- **Node and service monitoring**: Discovered Forms Listeners, Forms Servers, Load Balancer Servers, and Load Balancer Clients are monitored for the following events: Service down, Excessive memory usage, and Excessive CPU usage. When one of these events occurs, a pre-programmed action is taken to either alert the system administrator, or to try and fix the problem automatically.

13.3 OEM Components

There are three OEM components that you need to install in order to manage Forms Servers:

- **OEM Management Server (OMS)**: This is the software that controls and acts as the central repository for OEM. Install OMS on only one machine. This OMS machine will manage the other machines.
- **OEM Console**: This software provides the user interface for OMS.
- **OEM Agent**: This software collects Forms Server data and sends it back to OMS. The OEM Agent must be installed on every Forms Server machine that is to be managed by OMS.

13.4 Installing and Configuring OEM Components for Use with Forms

The OEM Management Server (OMS), OEM Console, and OEM Agent software are installed as part of Oracle Internet Application Server.

13.4.1 Configuring Forms Support for OEM

After Forms and OMS are installed, do the following. Be sure that the OMS service is not running while performing the following steps.

1. Change directories to $ORACLE_HOME\sysman\admin.
2. Connect to the database using a login that has system privileges.
3. Run the "createOEMFormsUser.sql" script to create an OEM Forms User who will support Forms specific data in the OEM repository. (You can modify this script to add default tablespace, quota, and so on. However, you cannot change the user name and password in the script.)
4. Connect to the database as the OEM Forms User. (See the SQL script you just ran for the user name and password.)
5. Run the "createOEMFormsTables.sql" script to create the necessary tables in the OEM repository.
6. Create a TNS entry in OMS Oracle Home under network/admin. The database name of the OEM Repository must exist in the tnsnames.ora file on the machine where you intend to run the OEM Console.

13.4.2 Starting the OMS Service

To start the OMS Service, type:

```
oemctrl start oms.
```

13.5 Managing Forms Servers from the OEM Console

You cannot manage a pre-existing Forms Listener from OEM. You must create it first from the OEM console.

13.5.1 Locating Nodes

Before OEM can manage a remote Forms Server machine, it has to locate it. To do this:

1. In the OEM Console, choose **Discover Node** from the menu.
2. Enter the node name. For example, dev2000srv-pc.

13.5.2 Entering the Administrative User's Credentials in the OEM Console

To enter the administrative user's credentials in the OEM console:

1. Start the OEM Console.
2. Choose **Preferences** from the System menu.

3. Choose the **Preferred Credentials** tab.
4. Find the name of the remote Forms Server machine you want to administer in the Service Name column. Be sure to select a row where the Service Type is Node.
5. Enter the name and password of the Administrative user created in.

13.5.3 Viewing Forms Runtime Instances from the OEM Console

To view Forms Runtime Instances from the OEM Console:

1. From the OEM Console, select **Developer Servers**, **Forms_Listeners_** <*RemoteMachineName*>, .
2. Right-click and select **List Runtime Processes**.

13.6 OEM Menu Options

The following menu options are available for managing Forms Listeners, Forms Servers, Load Balancer Servers, and Load Balancer Clients.

13.6.1 Controlling Forms Listeners Group

The commands available from the right mouse menu are:

- **Create New**: You are prompted for a list of parameters before a new listener process is created. Once the listener process is created, an entry is displayed in the Navigator tree, and the listener is started.
- **List Runtime Processes**: This will bring up a separate window with a list of Forms runtime processes running on this node. See Runtime Processes List Window.
- **Refresh**: This will discover existing Forms Listeners running on this node, and will also refresh the running / not running status of all Forms Listener instances on this node.

13.6.2 Controlling Forms Listeners Instance

The commands available from the right mouse menu are:

- **Start**: The listener starts, if the listener is currently down.
- **Stop**: The listener is shut down, and the Listener instance is marked as down with a special icon.
- **Create Like**: Much like a copy command, it creates another listener with the same parameters as the current one. You are prompted by a dialog similar to the Create New command to make any necessary changes.
- **Modify**: A dialog box allows you to modify the startup parameters and environment variables.
- **Delete**: The Listener instance is deleted from the navigator tree. A Forms Listener instance can only be deleted if there are no Runtime processes associated with this listener. A deleted listener is shut down from the node automatically.
- **Properties**: Brings up a list of parameters, environment variables, and runtime processes associated with this Forms Listener instance.

13.6.3 Runtime Processes List Window

This is a table type listing of all the current Forms Runtime processes on a particular node. Each row represents a Runtime process. The following fields are displayed:

- Listener name
- Node name
- IP address
- User
- PID
- Connect time
- Dynamic logging status
- Memory usage
- CPU %

13.6.4 Controlling Forms Runtime Processes

The commands available from the right mouse menu are:

- **Kill**: A kill signal is sent to the Runtime instance to stop its execution. This is mainly used to stop a malicious runtime process from doing further damage.
- **Logging ON**: Turns on dynamic logging for the Runtime instance. The log will be written to a temporary file with a generated file name. The file format is the same as the one generated by Forms Runtime Diagnostic (FRD).
- **Logging OFF**: Turns off dynamic logging for the Runtime instance.
- **View Log**: Displays the log file generated from the dynamic logging command.

13.6.5 Controlling Load Balancer Server Group

The command available from the right mouse menu is:

- **Create New**: A Load Balancer Server instance is created. Supported parameters are: *<port #1> <port #2> <max. no. of client> <trace level>*.

Load Balancer Server is also known as Metrics Server.

13.6.6 Controlling Load Balancer Server Instance

The commands available from the right mouse menu are:

- **Start**: Load Balancer Server is started.
- **Stop**: The server is shut down.
- **Create Like**: Much like a copy command, it creates another Load Balancer Server with the same parameters as the current one.
- **Modify**: A dialog box is displayed to allow you to modify the start up parameters and environment variables.
- **Delete**: Deletes Load Balancer Server from the Navigator tree. A deleted server is shut down from the node automatically.
- **Properties**: Bring up a separate window that shows any relevant information about this Load Balancer Server.

Load Balancer Server is also known as Metrics Server.

13.6.7 Controlling Load Balancer Client Group

The commands are exactly the same as the Load Balancer Server object type. The supported parameters are:

<Master Server host name> <Remote port> <Local port> <Scale Factor>

Load Balancer Client is also known as Metrics Client.

13.6.8 Controlling Load Balancer Client Instance

The commands are exactly the same as the Load Balancer Server object instance.

Load Balancer Client is also known as Metrics Client.

13.6.9 Monitoring Functions

Events are listed in the Events Management window of the OEM console. They can be turned on or off by registering or un-registering with OEM. Once an event is created and registered with OEM, OEM can notify the system administrator or run a *fixit* job when an event occurs.

The following events are available for you to register:

- **Listener down:** This event can be scheduled with or without a Listener *fixit* job. A Listener *fixit* job is available to restart the Listener when this event occurs.

Whenever a Listener goes down, an entry is written to the Event log, which is viewable from the OEM console.

Note: You must schedule a *fixit* job before you can schedule an event with a *fixit* job.

- **Load Balancer server down:** Similar to Listener down. This event can be scheduled with or without a Load Balancer server *fixit* job.
- **Load Balancer client down:** Similar to Load Balancer server down. This event can be scheduled with or without a Load Balancer client *fixit* job.
- **Excessive CPU usage by Runtime Process:** The system administrator is notified when a Runtime Process consumes too much CPU time. This event is checked every *X* seconds; you set the time interval. You can select an Alert threshold, Warning threshold, and the number of occurrences.
- **Excessive virtual memory usage by Runtime Process:** When virtual memory is consumed beyond a certain amount by a Runtime process, the system administrator is notified. This is similar to the Excessive CPU usage event. You can set the following parameters: event interval, warning threshold (in KB of virtual memory), alert threshold (in KB of virtual memory), and number of occurrences.

14

Capacity Planning Considerations

14.1 Introduction

This chapter explores Forms Server's scalability features. We researched the server's scalability by conducting a number of benchmark tests using popular hardware platforms and operating systems.

We measured these benchmarks:

- RAM per user
- Users per CPU

We got the following results for Forms Server 6.0:

For Windows NT:

Table 14–1 Benchmarks for Windows NT

Application size/complexity	RAM per user (MB)	Users per CPU
medium / moderate	2.5-6.0	100-300
small / simple	1.0-2.5	150-300

For Sun Solaris:

Table 14–2 Benchmarks for Sun Solaris

Application size/complexity	RAM per user (MB)	Users per CPU
medium / moderate	2.0-5.0	200-400
small / simple	1.0-2.0	300-500

Note: The results described in this chapter are specific to the 6*i* release of the Forms Server and should not be used for any of the previous releases of the product. The performance of this release has improved in comparison to earlier releases. This is due to a number of architectural and code optimizations, such as:

- Improved dynamic link library sharing under Windows NT
- Improved middle-tier record caching
- Improved messaging layer, thus reducing the overall processing on the server

Benchmark testing is an ongoing process at Oracle Corporation. The figures presented here represent the information available at the time of writing. Additional results will be published as they become available.

14.2 What Is Scalability?

Scalability is the ability to accommodate an increasing user population on a single system by adding more hardware resources to the system without changing the underlying software. A scalable system can accommodate the growing needs of an enterprise.

Choosing both hardware and software that can grow with your performance needs is a much better strategy than purchasing new software every time your performance needs change.

Consider these questions:

- How well does the application or operating system take advantage of additional system resources?
- How much memory do I need to support *n* number of users?
- Can I easily upgrade to a faster processor or multiple processors?
- How much incremental processing power does an additional processor provide?
- Are there additional features I can add later to boost performance (such as additional cache or a drive array controller)?

Answers to these questions depend heavily on the hardware, operating systems, and application software being used.

14.3 Criteria for Evaluating System Capacity

The scalability of a networked application is tied to the ability of the application server and the network topology to predictably accomodate an increase in user load.

It will be useful to understand the role of each component described in this section and how they can affect the overall scalability of a system, especially in a Forms Server environment.

This chapter uses as examples the two most commonly used server hardware and operating system combinations: Sun Solaris, running on Sun UltraSparc architecture, and Microsoft Windows NT, running on Intel architecture.

These areas are important in the evaluation of a Forms Server-based system:

- Processor
- Memory
- Network
- Shared Resources
- User Load
- Application Complexity

14.3.1 Processor

Work faster or work smarter? Processor technology has explored both paths. Typically, a company will release new generation architectures (working smarter) every two to three years. In between those releases, it will increase the processor speed (working faster). The speed of a processor, also called *clock speed*, is usually represented in megahertz (MHz). Processor speed is a good indication of how fast a computer system can run. Typically, computers used as servers employ more than one processor and are called multi-processor systems.

The metric that we are really interested in with regard to the Forms Server is the number of simultaneous users on each processor, sometimes called Users per Processor. This number will vary greatly for different types of processors. For an example of this variability, see Table 14–1 on page 14-1 and Table 14–2 on page 14-1.

From empirical data collected in the benchmark, a computer with a 400MHz Intel Pentium II Xeon processor with 1MB of L2 cache could support approximately twice as many users as compared to a 200MHz Pentium Pro system.

14.3.2 Memory

Memory is the amount of RAM that a computer system has available to launch and run programs. The amount of RAM in computer systems is usually represented in megabytes (MB).

In the normal execution of a program, the program is loaded in RAM, and the operating system swaps the program to disk whenever the program is inactive. The operating system brings the program back into RAM when it becomes active.

This activity is generally called *swapping*. Most operating systems, such as Sun Solaris or Microsoft Windows NT, perform swapping during normal operation. Swapping places additional demands on the processor. Excessive swapping tends to slow down a system considerably. To prevent slowed performance, include enough RAM in the server host machine.

The important metric is RAM that is required for every additional user that connects and runs an application via the Forms Server. This metric is also called the Memory per User. Often, performance-measuring tools do not provide an accurate measure of Memory per User. Study this metric carefully to determine memory requirements. For an example of memory per user, see Table 14–1 and Table 14–2 on page 14-1.

14.3.3 Network

In a multi-tier, Internet-based architecture such as Forms Server, the physical network that connects clients to the Forms Server and the connection between the Forms Server and the database are key factors in the overall scalability of the system. When you measure the performance of your Forms Server based system, pay careful attention to the performance of the physical network.

14.3.4 Shared Resources

The performance of an individual process in a multi-user, multi-process environment is directly proportional to the individual process' ability to be processed from main memory. That is, if required pages are swapped out to virtual memory in order to make room for other processes, performance will be impacted. One technique to increase the likelihood of finding the required page in main memory is to implement a shared memory model using Image Mapped Memory. Image Mapped Memory associates a file's contents in memory to a specific address space that is shared across processes.

Forms Server uses Image Mapped Memory. Individual Forms processes share a significant portion of the FMX file image, which reduces individual memory requirements and increases overall scalability.

14.3.5 User Load

In a benchmark scenario, it is impractical to configure a number of client machines (and users) that accurately represents a live application environment. In benchmarks, load simulators are used to simulate real users that perform transactions on the application server. The Oracle Tools Development Organization has developed a load simulator that mimics real-world Forms Server users by sending messages to the Server to simulate load. The load simulator is a small Java application that sits between the Forms Server and the UI client, intercepting the message traffic that passes between these two components.

Once event messages from the client are recorded, it is possible to play them back to the server. This simulates an actual user session. (Note that the UI client is not involved in playback mode.) During playback to the server, the load simulator is capable of playing back many user sessions. In this manner, the load simulator is able to calculate the total response time for a user by determining the total round trip time for messages between client and server. By summing the Total Response Time throughout an entire business transaction, it is possible to get a measurable metric for application performance.

14.3.6 Application Complexity

We performed tests against Forms applications of various complexity, from a simple single Form containing List of Values (LOVs) and pop-up windows, to complex applications containing multiple Forms and PL/SQL libraries (PLLs) open simultaneously. We tied application complexity to the number of modules that a user may be accessing at one time, rather than to the inherent complexity of any one module.

A good method for determining complexity is to look at all the dependencies appended to a Form. For example, a form may call other forms through the CALL_FORM or OPEN_FORM built-in. Additionally, it may have attached menus (MMX files) and load external business logic through the use of PL/SQL libraries (PLL files). All of these factors contribute to memory usage per user.

The following table classifies the level of complexity of Oracle Forms applications.

Table 14–3 Determining Application Complexity

Application size/complexity	Total size of concurrent modules in memory
large / complex	> 10MB
medium / moderate	2MB – 10MB
small / simple	< 2MB

We tested two applications of different complexity:

- The first was a simple Customer-Order-Entry screen that contained appropriate menus and List of Values. Only a single form was active at any given time.
- The second was a moderate to complex application. We used an actual customer application, a help desk and customer support system. This application had numerous modules open simultaneously and complex business logic within individual modules.

To represent a realistic user community, that is, one where there is a mixed workload, the test encompassed a number of transactions that mimicked the activities performed by a Service Desk Clerk in a 45-minute scenario.

Step by step definition of the tasks implemented.

Step	Task performed
1	Launch Service View Application – Login
2	[NAVIGATE] to Notifications Screen
3	[CALL] Progressions Screen Transactions: Enter a Parameterized query
4	[OPEN] Problem Screen [NAVIGATE] to the various Tabs (PL / SQL execution) [NAVIGATE] to all the fields on the screen
5	[CALL] Services Screen Transactions: Enter a Blind Query [NAVIGATE] to all the queried records
6	[REPEAT] Scenario 2 – 5

14.4 Determining Scalability Thresholds

To get a feel for the decrease in performance with increasing user loads, it is first necessary to determine the time taken by a given user to perform a given application task. This Total Response Time metric differs from merely testing response time for a given physical transaction or network round trip. It looks at the total time taken (by an average user) to perform the business task at hand (that is, the sum of all interactions with the Forms Server and database that takes place as part of the business transaction).

To gain some empirical information about overall system resources, the scalability testing also uses the native operating system monitoring utilities (such as Windows NT performance monitor) to determine values for both physical and virtual memory usage, and for total CPU utilization.

By using the Total Response Time metric with the empirical measurements, it was possible to determine the point at which, given an increasing user load, performance for a given user significantly degraded. Having determined the number of users that can be supported with acceptable performance, individual memory consumption becomes a simple equation of the total memory available divided by the number of users accessing the application.

For example:

On a given hardware platform with 512MB of RAM, performance is constant for up to 60 concurrent users. Then it degrades significantly. From this, we can specify that the maximum number of users supported is 60.

Allowing for a nominal operating system overhead (~32MB), individual memory usage would be (512-32) / 60 or 8MB per user.

14.5 Sample Benchmark Results

The following sections define the systems we tested, the results of the tests, and a brief analysis for the following scenarios:

- Medium-Complex Application on a Low-Cost Intel Pentium-Based System
- Medium-Complex Application on an Intel Pentium II Xeon-Based System
- Medium-Complex Application on an Entry-Level Sun UltraSparc Server
- Simple Application on an Intel Pentium II Xeon-Based System
- Simple Application on an Entry-Level Sun UltraSparc Server

14.5.1 Medium-Complex Application on a Low-Cost Intel Pentium-Based System

Parameters:

Application size/complexity	CPU	RAM	Operating System	Swap
medium (between 2MB and 10MB)	2-200 MHz Pentium Pro	512MB	Windows NT 4.0 Server (SP 3)	2GB

Results:

Users per CPU	Memory per user
100	2.4MB

Analysis:

This system is one of the cheapest systems we used to test the scalability of a medium-complexity application. The system could handle about 200 users very efficiently. Performance degraded dramatically beyond 200 users. This system is cost effective as a small departmental server for up to 200 users with applications that fall in the medium complexity class.

14.5.2 Medium-Complex Application on an Intel Pentium II Xeon-Based System

Parameters:

Application size/complexity	CPU	RAM	Operating System	Swap
medium (between 2MB and 10MB)	2-400 MHz Pentium II Xeon with 1MB L2 cache	512MB	Windows NT 4.0 Server (SP 3)	2GB

Results:

Users per CPU	Memory per user
200	1.2MB

Analysis:

This system is one of the newest Intel Pentium II Xeon based servers we used to test the scalability of a medium complexity application. The system handled about 400 users very efficiently. Performance degraded dramatically beyond 400 users. The system is cost-effective as a large departmental server or as an entry-level Enterprise Server for small to medium businesses.

14.5.3 Medium-Complex Application on an Entry-Level Sun UltraSparc Server

Parameters:

Application size/complexity	CPU	RAM	Operating System	Swap
medium	2-248 MHz Ultra Sparc	512MB	Solaris 2.5.1	2GB

Results:

Users per CPU	Memory per user
200	1.3MB

Analysis:

The system handled about 375 users very efficiently. Performance degraded dramatically beyond 375 users. The system seemed to slow down due to excessive paging and swapping activity, which indicates that the real bottleneck was physical

memory. This system is cost-effective for larger departments or small to medium businesses running medium-complexity applications.

14.5.4 Simple Application on an Intel Pentium II Xeon-Based System

Parameters:

Application size/complexity	CPU	RAM	Operating System	Swap
small (less than 2MB)	2-400 MHz Pentium II Xeon with 1MB L2 cache	512MB	Windows NT Server 4.0 (SP 3)	2GB

Results:

Users per CPU	Memory per user
250	1MB

Analysis:

The Pentium II Xeon based server handled about 500 users very efficiently with a small application.

14.5.5 Simple Application on an Entry-Level Sun UltraSparc Server

Parameters:

Application size/complexity	CPU	RAM	Operating System	Swap
small (less than 2MB)	2-248 MHz Ultra Sparc	512MB	Solaris 2.5.1	2GB

Results:

Users per CPU	Memory per user
240	1MB

Analysis:

This system is an entry-level Sun Ultra Sparc System. The system handled about 480 users very efficiently. Performance degraded dramatically beyond 480 users.

15

Troubleshooting Solutions

15.1 Introduction

This chapter contains information about troubleshooting solutions for the Forms Server in the following sections:

- Checking the Status of the Forms Server
- Starting the Forms Server
- Stopping the Forms Server Process
- Starting the Forms Server Log
- Troubleshooting FAQ

15.2 Checking the Status of the Forms Server

To check the status of the Forms Server:

On Microsoft Windows NT:

1. Press **Control**+**Alt**+**Delete** to display the Windows NT Security dialog.
2. Choose **Task Manager**.
3. In the Task Manager, click the **Processes** tab.

If a server process is running, the Task Manager will display a process called IFSRV60.EXE, and multiple occurrences of a process called IFWEB60.EXE (one for every active connection).

On UNIX:

At the UNIX prompt, type: `ps -ef | grep f60srvm` and press **Enter**.

A list of process IDs will appear on the screen. If the Listener is running, the list will include a process called f60srvm, and multiple occurrences of the f60webm process. (There is one process for every active connection, plus one spare connection ready for the next user if the default value of *pool* is being used. If *pool* is set to 5, there will be 5 spare connections.)

15.3 Starting the Forms Server

To start the Forms Server:

As a service on Microsoft Windows NT:

You can remove an existing Forms Server service and reinstall it using new start-up parameters.

1. From a command window, type the following:

```
ifsrv60 -remove <FormsServerServiceNameToBeRemoved>
```

2. Type the following:

```
ifsrv60 -install <NewFormsServerServiceName> port=<portNum>
mode=<socket/http/https> [pool=<numOfRunforms> log=<logfilePath>
exe=<RunformexeName>]
```

3. Press **Enter**. A server process starts running on the specified port number.

See Section 5.4, "Description of Forms Server Startup Parameters" for startup parameter definitions.

In console mode on Microsoft Windows NT:

1. On the taskbar, choose **Start → Run**.
2. Type:

```
<ORACLE_HOME>\bin\ifsrv60 <FormsServerName> port=<portNum>
mode=<socket/http/https> [pool=<numOfRunforms> log=<logfilePath>
exe=<RunformexeName>]
```

3. Press **Enter**. A server process starts running on the specified port number.

See Section 5.4, "Description of Forms Server Startup Parameters" for start-up parameter definitions.

On UNIX:

1. From the UNIX prompt, type:

```
cd <ORACLE_HOME>.
```

2. Press **Enter**.
3. Type

```
forms60_server start
```

4. Press **Enter**. The server starts running in the background.

See Section 5.4, "Description of Forms Server Startup Parameters" for start-up parameter definitions.

15.4 Stopping the Forms Server Process

To stop the Forms Server process:

As an NT service on Microsoft Windows NT:

1. Go to the Control Panel, and select **Services**.
2. Locate and select the Forms Server process.
3. Click **Stop**.

In console mode on Microsoft Windows NT:

1. Check the status of the Forms Server. If the server is running, the Task Manager will display a process called IFSRV60.EXE.
2. Select IFSRV60.EXE, and click **End Process**.

On UNIX:

1. Check the status of the Forms Server. A list of process IDs will appear on the screen. Note the process ID for the f60srvm process.
2. At the UNIX prompt, type

```
kill process_ID
```

or type

```
kill -g
```

3. Press **Enter**.

15.5 Starting the Forms Server Log

The Forms Server will create a log file if you start the server using the log option, as follows:

```
ifsrv60 -install Forms60Server log=<\PathName\LogFileName> port=<portNum>
mode=<socket/http/https>
```

The log contains diagnostic information.

15.6 Troubleshooting FAQ

Problem	Solution
You cannot run Web-enabled Forms applications with a non-Java-enabled Web browser.	If you are not sure your Web browser is Java-enabled, check your Web browser's network preferences. The Enable Java and Enable JavaScripts check boxes must be checked.
You see the error message "Cannot bind to port 9000" when you try to start the Forms Server.	Another process may be using the port. It could be another occurrence of the Forms Server; check that the Forms Server is not already running. If you just stopped the Forms Server, it may take a minute or two for existing connections to port 9000 to reopen.
The Forms Client does not download to your Web browser.	Check that you have defined a virtual directory to point to the Oracle Java class files (codebase).
The server will not allow the client to connect, although all connection data is correct.	If the server is using 128-bit encryption (domestic license) and the client cannot support this (because it uses 40-bit encryption under an export license), check the FORMS60_HTTPS_NEGOTIATE_ DOWN environment variable. If this variable is set to FALSE, the server will reject client connection requests. If needed, check the Java console and the server log file (if one is available) to see the level of encryption being used by th client and server.
The Forms Server seems to ignore the user ID, password, and database SID parameter values you pass in your application base HTML file.	Make sure you preface the values with the parameter "userid=". For example: userid=scott/tiger@inventory
The Forms Server seems to not pick up your variable changes.	Stop and restart the Forms Server.

Problem	Solution
You experience problems when using a security firewall, and you are using a proxy server to access anything outside the firewall.	Make sure your proxies are set to manual configuration.
The HTML page and applet download at startup, and the applet starts running, but nothing else seems to happen.	Check the following: First, ensure that the Forms Client indeed is running; if it is, you should see a message in the status bar of your Web browser: applet oracle.forms.engine.Main running. If you see this message, but your application still does not appear, check the following: 1. Make sure the Forms Server and your Web server are installed on the same application server. Due to a current Java restriction, they must be installed on the same server. 2. Check your application base HTML file and configuration file to make sure you specified a valid directory path and filename for the .FMX file. You must use a physical directory path, not a virtual directory path. 3. Try setting a preference in your Web browser to display the Java console. This allows you to view runtime Java error messages.
Applet not able to connect to Forms Server.	Make sure that the "mode" setting on the server matches the "connectionType" in the base HTML file.
You experience trouble connecting to a local database.	It could be a result of the following: * If you do not specify a Net8 v2 connect string, you will receive errors. The Forms Server runtime engine will not accept connect strings of type LOCAL, TWO_TASK, and so on. * If you are using a Net8 v2 connect string and you still cannot connect to the database, make sure the Forms Server is running; on most installations, the Server is not automatically restarted after a reboot. * You must have the valid connect string in the TNSNAMES.ORA file on your application server, not on your client machine. The application logic is running on an application server, not on users' client machines.
You experience unpredictable behavior after modifying the CLASSPATH environment variable.	Changing the setting of the CLASSPATH environment variable on your application server or on a user's machine can produce unpredictable results. Setting the variable to a directory that overlaps with the directory tree where Forms Java class files are located can cause filename overlap.

Problem	Solution
There appear to be several unused processes running on the server.	Recall that for each user running a Web-enabled Form Builder application, a Forms Server runtime process (ifweb60.exe and ifsrv60 on Windows; f60webm and f60srvm on UNIX) starts on your application server. Each runtime process should end when the user exits the application. The process will remain on the server if a user exits the browser without cleanly exiting the application. To cleanly exit the application, use the menu or the [Exit/Cancel] key function, then exit the browser.

Part II

Appendices

A

Forms Server Parameters

A.1 Introduction

This appendix contains the parameters you use to configure Forms Server.

A.2 Windows 95 and Windows NT Registry

For Windows 95 and Windows NT, the Oracle Univerasal Installer creates a new ORACLE section in your registry. The Oracle registry contains configuration parameters that control such things as the name of the Oracle home directory, the location of the product preference file, and the location of the help files. If you use Net8 for Windows, the configuration parameters also determine the driver to be used for network communications and the values that Net8 should use for its operating parameters.

A.2.1 Viewing and Modifying the Registry

You can view and optionally edit the Microsoft Windows Registry with the Registry Editor. This editor is located in the directory where your Windows software is installed.

To start the editor:

1. Choose **Start → Run**.
2. Type `REGEDIT`.
3. Click **OK**.
4. In the Registry Editor, expand the HKEY_LOCAL_MACHINE node.
5. Expand the SOFTWARE node.
6. Click the ORACLE key to display the Oracle configuration parameters.

7. You can modify any parameter value by double-clicking the parameter name to display the Edit String dialog.
8. Change the value in the Value data text box.
9. Click **OK** to accept the new value.

A.3 Configuration Parameters

The Oracle Installer automatically sets many parameters. Some of the parameters are required by Oracle products, and are listed in Table A–1. Other parameters allow you to customize product behavior. They are described in Section A.3.2, "Customizable Parameters".

A.3.1 Required Parameters

The parameters listed in this section are automatically set or removed by the Oracle Installer. They are required by various Oracle products to function properly.

Caution: Do not change the settings of parameters listed in this section. Doing so may cause one or more Oracle products to stop functioning correctly.

The appearance of *nn* in the parameters listed below specifies a product or component release number. This number *may* change when you upgrade to a new release of an Oracle product.

Table A–1 Required parameters

Parameter	Setting
BROWSER*nn*	*ORACLE_HOME*\BROWSE*nn*
DE*nn*	*ORACLE_HOME*\TOOLS\COMMON*nn*
FORMS*nn*	*ORACLE_HOME*\FORMS*nn*
GRAPHICS*nn*	*ORACLE_HOME*\GRAPH*nn*
MM*nn*	*ORACLE_HOME*\TOOLS\COMMON*nn*
OCL*nn*	*ORACLE_HOME*\GRAPH*nn*
PRO*nn*	*ORACLE_HOME*\PRO*nn*
RDBMS*nn*	*ORACLE_HOME*\RDBMS*nn*
RW*nn*	*ORACLE_HOME*\REPORT*nn*
TK*nn*	*ORACLE_HOME*\TOOLS\COMMON*nn*

Table A–1 Required parameters

Parameter	Setting
BROWSER*nn*	*ORACLE_HOME*\BROWSE*nn*
VGS*nn*	*ORACLE_HOME*\TOOLS\COMMON*nn*

A.3.2 Customizable Parameters

The parameters listed in this section control various aspects of your Oracle products. You may change the settings of these parameters to customize behavior.

The sections below list the default setting (if any) of each parameter. Parameters that are not automatically set with default values are noted. The parameter listings include descriptions of valid values and examples.

FORMS60_PATH

Default: *ORACLE_HOME*\FORMS60\PLSQLLIB

Valid Values: any directory on any drive

Example:

FORMS60_PATH=C:\oracle\apps\forms;C:\myfiles

This parameter specifies the search path for files used in a Form Builder runtime application. These include form files (.fmx), menu files (.mmx), PL/SQL libraries (.pll), and other objects that the application attempts to load from a file at runtime. For example, if you import the image file scooter.tif, Form Builder searches in the directories specified by FORMS60_PATH to find that file.

FORMS60_PATH can specify multiple directories. Use a semicolon (;) to separate directory names in a list of paths.

FORMS60_REPFORMAT

Default: none

Valid Values: HTML, PDF

Example:

FORMS60_REPFORMAT=HTML

If you are invoking a browser to run a report from a form via RUN_PRODUCT, you must set the FORMS60_REPFORMAT environment variable. This parameter specifies the report format.

FORMS60_TIMEOUT

Default: 15

Valid Values: 1 – 1440 (1 day)

Example:

FORMS60_TIMEOUT=1440

This parameter specifies the amount of time in elapsed minutes before the Forms Server process is terminated when there is no client communication with the Forms Server.

GRAPHICS60_PATH

Default: none

Valid Values: any directory on any drive

Example:

GRAPHICS60_PATH=C:\oracle\apps\graphics;C:\myfiles

This parameter specifies the search path for files used in a Graphics runtime application. These include display files (.ogr), images, external queries, and other objects that the application attempts to load from a file at runtime. For example, if you import the image file scooter.tif, Graphics Builder searches in the directories specified by GRAPHICS60_PATH to find that file.

GRAPHICS60_PATH can specify multiple directories. Use a backslash (\) to separate directories in a path, and a semicolon (;) to separate complete paths.

NLS_LANG

Default: AMERICAN_AMERICA.WE8ISO8859P1

Valid Values: See the *NLS Reference Manual* for a current list of available values, or see the following file on your CD: \bonus\nls\nlsd2r1.wri

Example:

NLS_LANG=AMERICAN_AMERICA.WE8ISO8859P1

This parameter sets the language in which message files appear. The syntax for NLS_LANG is as follows:

NLS_LANG=<*language*>_<*territory*>.<*char_set*>

Where:

- *Language* specifies the language and its conventions for displaying messages and day and month names.
- *Territory* specifies the territory and its conventions for calculating week and day numbers.
- *Char_set* specifies the character set used for the UPPER, LOWER, and INITCAP functions, and the type of sort used by an ORDER BY query. This argument also controls the character set used for displaying messages.

ORACLE_HOME

Default: C:\ORAWIN95 on Window95 or C:\ORANT on Windows NT

Valid Values: any directory on any drive

Example:

ORACLE_HOME=C:\orawin95

This parameter specifies the home directory in which Windows Oracle products are installed. This directory is the top directory in the Oracle directory hierarchy.

B

Oracle JInitiator

B.1 Introduction

This appendix describes the benefits of using Oracle JInitiator as a plug-in for your users' Web browsers. Oracle JInitiator makes it possible for users to run Forms Server applications using Netscape Navigator or Internet Explorer. It provides the ability to specify the use of a specific Java Virtual Machine (JVM) on the client, rather than using the browser's default JVM.

Oracle JInitiator runs as a plug-in for Netscape Navigator and as an ActiveX component for Internet Explorer. Oracle JInitiator does not replace or modify the default JVM provided by the browser. Rather, it provides an alternative JVM in the form of a plug-in.

B.1.1 Why Use Oracle JInitiator?

Oracle JInitiator delivers a certified, supportable, Java Runtime Environment (JRE) to client desktops, which can be launched transparently through a Web browser.

Oracle JInitiator is Oracle's version of JavaSoft's Java Plug-in. The JavaSoft Plug-in is a delivery mechanism for a JavaSoft JRE, which can be launched from within a browser. Likewise, Oracle JInitiator is providing a delivery mechanism for an Oracle certified JRE, which enables Forms Developer applications to be run from within a browser in a stable and supported manner.

In addition to providing a certified platform for the execution of Forms Developer applications, Oracle JInitiator provides a number of additional features over and above the standard JavaSoft Java Plug-in. These include JAR file caching, incremental JAR file loading, and applet caching.

B.1.2 Benefits of Oracle JInitiator

Oracle JInitiator provides these benefits:

- It allows the latest Oracle-certified JVM to run in older browser releases.
- It ensures a consistent JVM between different browsers.
- It is a reliable deployment platform. JInitiator has been thoroughly tested and certified for use with Forms Server.
- It is a high-performance deployment environment. Application class files are automatically cached by JInitiator, which provides fast application start-up.
- It is a self-installing, self-maintaining deployment environment. JInitiator automatically installs and updates itself like a plug-in or an Active-X component. Locally cached application class files are automatically updated from the application server.

B.2 Using Oracle JInitiator

The first time the client browser encounters an HTML file that specifies the use of Oracle JInitiator, it is automatically downloaded to a client machine from the application server. It enables users to run Forms and Graphics applications directly within Netscape Navigator or Internet Explorer on the Windows 95 and Windows NT 4.0 platforms.

The installation and updating of Oracle JInitiator is performed using the standard plug-in mechanism provided by the browser. Oracle JInitiator installation performs the required steps to run Forms Developer applications as trusted applets in the Oracle JInitiator environment.

B.2.1 Supported Configurations

Oracle JInitiator supports the following configurations:

	Internet Explorer 4.0	Internet Explorer 5.0	Navigator 4.0	Navigator 4.5
Windows 95	X	X	X	X
Windows NT	X	X	X	X

B.2.2 System Requirements

The minimum system requirements for Oracle JInitiator are:

- Windows 95 or Windows NT 4.0
- Pentium 90 mHz or better processor
- 12MB free hard disk space (recommended 20MB)
- 16MB system RAM (recommended 24MB)

B.2.3 Using Oracle JInitiator with Netscape Navigator

Oracle JInitiator leverages the Netscape Navigator plug-in architecture in order to run inside the browser in the same way other plug-ins, such as QuickTime movies or Shockwave animations operate. Using the Netscape HTML <EMBED> tag, Web application developers can specify that plug-ins run as part of a Web page. This is what makes it possible for Oracle JInitiator to run inside the Web browser with minimal user intervention.

When Navigator first encounters an HTML page that specifies the use of Oracle JInitiator, users will see a "Plug-in Not Loaded" dialog on the HTML page, which directs the user to the Oracle JInitiator download page. Users can then download the version of Oracle JInitiator for their operating system and install it.

Once Oracle JInitiator is installed, users must shut down Navigator, restart it, and then revisit the original HTML page. Oracle JInitiator will then run and use the parameters in the <EMBED> tag to render the applet. The next time Navigator encounters a Web page that specifies Oracle JInitiator, Navigator will seamlessly load and run the plug-in from the local disk, without user intervention.

B.2.4 Using Oracle JInitiator with Microsoft Internet Explorer

Oracle JInitiator leverages the Microsoft Internet Explorer extension mechanism for downloading and caching ActiveX controls and COM components. Using the HTML <OBJECT> tag, Web application developers can specify that ActiveX controls or COM components should run as part of a Web page. Such components include Oracle JInitiator.

When Internet Explorer first encounters an HTML file that has been modified to specify the use of Oracle JInitiator, Internet Explorer will ask the user if it is okay to download an ActiveX control signed with a VeriSign digital signature by Oracle Corporation. If the user clicks "Yes," Internet Explorer will begin downloading Oracle JInitiator. Oracle JInitiator will then run and use its parameters in the

<OBJECT> tag to render the applet. The next time Internet Explorer encounters a Web page modified to support Oracle JInitiator, it will seamlessly load and run Oracle JInitiator from the local disk, without user intervention.

B.2.5 Setting up the Oracle JInitator Plug-in

To set up the Oracle JInitiator plug-in:

- Add Oracle JInitiator HTML markup to your base HTML file.
- Install Oracle JInitiator on your server (for server-based testing purposes only).
- Customize the Oracle JInitiator download file.
- Make Oracle JInitiator available for download.

B.2.5.1 Adding Oracle JInitiator Markup to Your Base HTML File

To add Oracle JInitiator markup to your base HTML file:

1. Open your base HTML file within a text editor.
2. Add the OBJECT and EMBED tags.

   For examples of added markup, refer to Section B.2.7, "Oracle JInitiator tags for a base HTML file".

B.2.5.2 Installing Oracle JInitiator on your Web Server

Installing the Oracle JInitiator on your server allows you to test and refine your configuration before deploying your applications to customers. Note that this is not a required step. It is merely useful for local system testing.

To install Oracle JInitiator on your Web server:

1. Double-click jinit11711.EXE.
2. Follow the installation instructions.

B.2.5.3 Customizing the Oracle JInitiator Download File

The Oracle JInitiator download file (JINIT_DOWNLOAD.HTM) is the template HTML file that allows your users to download the Oracle JInitiator file.

To customize the Oracle JInitiator download file:

1. Open the JINIT_DOWNLOAD.HTM file within an HTML or text editor.
2. Modify the text as desired.
3. Save your changes.

B.2.5.4 Making Oracle JInitiator available for download

To make Oracle JInitiator available for download:

1. Copy jinit11*x*.EXE to your Web server.

   You must copy jinit11*x*.EXE to the location that was specified within the base HTML file.

2. Copy JINIT_DOWNLOAD.HTM to your Web server.

   You must copy JINIT_DOWNLOAD.HTM to the location that was specified within the base HTML file.

B.2.6 Modifying the Oracle JInitiator plug-in

To modify the Oracle JInitiator plug-in:

- Modify the cache size for Oracle JInitiator.
- Modify the heap size for Oracle JInitiator.
- View Oracle JInitiator output.

B.2.6.1 Modifying the cache size for Oracle JInitiator

To modify the cache size for Oracle JInitiator:

1. From the Start menu, choose **Start → Programs → Oracle JInitiator → Control Panel**.
2. Click the **Basic** tab.
3. In the Java Run Time Parameters field, specify the Dcache size. For example, specifying Dcache.size=20000000 sets the cache size to 20MB.

The default cache size for Oracle JInitiator is 20000000. This is set for you when you install Oracle JInitiator.

B.2.6.2 Modifying the heap size for Oracle JInitiator

To modify the heap size for Oracle JInitiator:

1. From the Start menu, choose **Start → Programs → Oracle JInitiator → Control Panel**.
2. Click the **Basic** tab.
3. In the Java Run Time Parameters field, specify the mx size. For example, specifying mx64m means setting maximum heap size to 64MB.

The default maximum heap size for Oracle JInitiator is 64MB. This has been set for you when you install Oracle JInitiator.

B.2.6.3 Viewing Oracle JInitiator output

To view Oracle JInitiator output:

1. From the Start menu, choose **Start → Programs → Oracle JInitiator → Control Panel**.
2. Click the **Basic** tab.
3. Check the **Show Java Console** check box to enable debug output.

B.2.7 Oracle JInitiator tags for a base HTML file

This example illustrates the Oracle JInitiator markup for both Microsoft Internet Explorer and Netscape Navigator. Adding these tags to your base HTML file will enable your applications to run within both Netscape and Microsoft browsers.

```
<HTML>
<BODY>
<P>
<OBJECT classid="clsid:9F77a997-F0F3-11d1-9195-00C04FC990DC"
WIDTH=600
HEIGHT=480
codebase="http://acme.com/jinit11711.exe#Version=1,1,7,11">
<PARAM NAME="CODE" VALUE="oracle.forms.engine.Main" >
<PARAM NAME="CODEBASE" VALUE="/forms60code/" >
<PARAM NAME="ARCHIVE" VALUE="/forms60code/f60all.jar" >
<PARAM NAME="type" VALUE="application/x-jinit-applet;version=1.1.7.11">
<PARAM NAME="serverPort" VALUE="9000">
<PARAM NAME="serverArgs" VALUE="module=order.fmx">
<PARAM NAME="serverApp" VALUE="default">
<COMMENT>
<EMBED type="application/x-jinit-applet;version=1.1.7.11"
java_CODE="oracle.forms.engine.Main"
java_CODEBASE="/forms60code/"
java_ARCHIVE="/forms60code/f60all.jar"
WIDTH=600
HEIGHT=480
serverPort="9000"
serverArgs="module=order.fmx"
serverApp="default"
pluginspage="http://acme.com/jinit_download.htm">
<NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>
</BODY>
</HTML>
```

B.3 Oracle JInitiator FAQ

The most frequently asked questions about Oracle JInitiator are discussed in detail in the following sections:

- Certification and Availability
- Support
- Installation
- Operation of Oracle JInitiator
- Caching

B.3.1 Certification and Availability

When will Oracle JInitiator be available?

Oracle JInitiator has been available since September 1998 for the deployment of custom Forms Developer applications. Oracle Applications completed certification of Oracle JInitiator in February 1999.

How is Oracle JInitiator distributed?

Starting with release 6*i* of Forms Developer, Oracle JInitiator will be shipped as part of the Forms Developer distribution CD. Oracle JInitiator is also available for download from the Forms Developer section of the Oracle Web site: http://www.oracle.com/tools/dev_server. Updates for Oracle JInitiator may also be obtained through the Oracle Worldwide Support Organization.

Will Oracle JInitiator work on non-Windows platforms?

Oracle has no current plans for porting Oracle JInitiator to non-Microsoft Windows platforms. However, we are working very closely with a number of hardware vendors to provide support and certification for running Forms Developer applications on non-Microsoft Windows platforms.

What versions of Netscape Navigator and Internet Explorer is Oracle JInitiator certified with?

Oracle JInitiator will be certified with the latest production releases of these browsers when each Oracle JInitiator release undergoes final QA testing. Oracle will also be providing support for earlier releases of the browsers. The exact browser versions that have been certified will be contained in the accompanying documentation for an Oracle JInitiator release.

What is the difference between the JavaSoft Java Plug-in and Oracle JInitiator?

The primary difference is that Oracle JInitiator includes the Oracle certified JRE whereas the JavaSoft Java Plug-in is shipped with a JavaSoft JDK reference implementation. JavaSoft's implementation has not been certified with Forms Developer applications. Forms Developer places extreme demands on the JRE; so we have modified JavaSoft's JRE to perform under extreme conditions.

While Oracle is diligent in notifying JavaSoft of its enhancements, it is not possible to wait until JavaSoft can provide a new version with the included enhancements.

The JavaSoft Plug-in is a delivery mechanism for a JavaSoft JRE which can be launched from within a browser. Likewise, Oracle JInitiator is providing a delivery mechanism for an Oracle certified JRE, which enables Forms Developer applications to run within a browser in a stable and supported manner.

Since Oracle is responsible for the production of Oracle JInitiator, we provide full product support for it. Through the Oracle World Wide Support Organization, Oracle customers can obtain the relevant level of support required to support their applications.

In addition to providing a certified platform for the execution of Forms Developer applications, Oracle JInitiator provides a number of additional features over and above the standard JavaSoft Java Plug-in. These features include JAR file caching, incremental JAR file loading, and applet caching.

Why is Oracle certifying and delivering a specific JRE rather than using the JRE provided by JavaSoft?

Forms Developer has responded to its customers who are moving to server-based deployment as a way to reduce computing costs, but also realize the need to protect their investment in existing applications that are essential to their business.

Providing our customers with the ability to run their existing applications completely unchanged on a Java platform places unique demands on Java, especially given that many of these applications are large and complex.

Can the JavaSoft Java Plug-In be used to run Forms Developer applications?

Using the JavaSoft Plug-In to deploy Forms Developer applications has not been certified and is therefore not a supported deployment configuration. Today, the JRE provided by Oracle JInitiator includes a number of enhancements that are not yet available in the JRE provided by JavaSoft. In addition, Oracle is able to provide full support for the Oracle JInitiator through the Oracle Worldwide Support Organization.

Does Oracle intend to support native browser deployment?

The primary problem with providing native browser support is the dependence on browser vendors and platform providers to support the same version and quality level of Java that is required by Forms Server. This dependency has prevented Oracle from certifying native browser deployment as a deployment option in the timeframe that our customers require. Therefore, we are fully endorsing Oracle JInitiator as our Internet application deployment strategy. This ensures a stable and supported platform on which to deploy Forms Server applications.

B.3.2 Support

Who will provide support for Oracle JInitiator?

Oracle Corporation provides full support for Oracle JInitiator through the Oracle Worldwide Support Organization.

Which versions of Forms Developer Server does Oracle JInitiator support?

Oracle will support Forms Server Release 1.6 and later with Oracle JInitiator running on the client.

Is Oracle JInitiator supported with Oracle Applications?

Yes. The Oracle Applications group has certified the use of Oracle JInitiator for the running of Oracle Applications within Netscape Navigator 4.06 and later and Microsoft Internet Explorer 4.0 and later.

B.3.3 Installation

What do I need to install on the client in order to run Forms Developer applications in the Web browser?

By leveraging the standard browser extension mechanisms provided by both Netscape Navigator and Microsoft Internet Explorer, Oracle JInitiator is able to automatically download itself to the client machine when the browser first encounters an HTML page that requires it. Oracle JInitiator is then installed using the method required for the addition of Plug-ins or ActiveX Objects by the browser currently in use.

How large is Oracle JInitiator when it is downloaded to the client?

The compressed Oracle JInitiator distribution is approximately 8MB and expands to approximately 10MB when completely installed on the client.

Is it possible to perform a silent installation of Oracle JInitiator where the user does not have to enter any details?

Oracle JInitiator supports a silent installation mode in which the user doesn't need to actively step through the installation process provided by the InstallShield. To perform the silent installation, the user must download the Oracle JInitiator distribution to their machine and then specify "`-s -sm`" from the command line or from the Windows Run dialog when running the downloaded executable.

For example to perform a silent installation from the command line, the user would open a DOS shell and type:

```
C:\TEMP> jinit1179 -s -sm
```

To perform a silent installation using the Windows Run dialog, the user would click **Start → Run** and then enter `jinit1179 -s -sm` in the Run dialog window that appears:

Is it possible to perform the Oracle JInitiator installation from a central server such that user interaction is not required?

Using the facilities provided by the host operating systems, it is possible to install Oracle JInitiator on each client desktop without user intervention. This involves the System Administrator accessing each client machine and running the silent, non-GUI installation option of Oracle JInitiator.

Can I force Oracle JInitiator to use the same configurations for Proxy Servers, etc. as the browser in which it is running?

The operation of Oracle JInitiator is controlled via the Oracle JInitiator Control Panel. The Oracle JInitiator Control Panel is installed at the same time Oracle JInitiator is installed and can be accessed from the **Start → Programs** menu.

With the Oracle JInitiator Control Panel, you can configure Oracle JInitiator to use either its own specific Proxy settings or the defaults supplied by the browser from which it is invoked. Select the **Proxies** tab and insert the appropriate settings.

How can I force my browser clients to download and install a new version of Oracle JInitiator?

Oracle JInitiator functions as a Netscape Plug-in or a Microsoft ActiveX object depending on the type of browser being used. The browser uses a MIME type to provide a mapping between an HTML page request and the required

Plug-in / ActiveX object. Each Oracle JInitiator installation has a specific MIME type associated with it. When a browser loads an HTML page that contains a MIME type that it is not aware of, the browser informs the user that it does not have the required Plug-in / ActiveX object and will open a dialog that will help the user retrieve it.

By changing the MIME type specified in your application's HTML page to be a later version, the browser will detect that it does not have a valid Plug-in / ActiveX object for that MIME type and will prompt the user to download a new file so it can serve the request completely.

For example:

An HTML page HR.HTML allows users to run the HR application. The HR.HTML page indicates to the browser that it should use Oracle JInitiator version 1.1.5.21.1 through the MIME type value.

If a later release of Oracle JInitiator is obtained and placed on the server, the client browser can be forced to use the newer version by modifying the version specific lines in the HR.HTML file with the newer version release information.

I pressed the Cancel button on the Netscape "Plug-in Not Loaded" dialog and now I never get prompted to install Oracle JInitiator. How do I install the Plug-in?

Netscape uses the Windows registry to store information about installed Plug-ins. As soon as the "Plug-in Not Loaded" dialog appears, Netscape writes the details for the Plug-in into the registry, irrespective of whether the Plug-in is actually installed or not. When a page is encountered that calls for the use of that specific Plug-in, it will appear to Netscape that the Plug-in is installed because the registry says it was. This results in the "Plug-in Not Loaded" dialog box not being shown again. To overcome this, you can force Netscape to load a Plug-in by clicking the Plug-in missing icon. This will result in Netscape displaying the Plug-in download dialog.

I have a lot of HTML pages that have different MIME types in them. Will the latest Oracle JInitiator release still run with these earlier MIME types?

Currently the Netscape browser has limit of 256 characters that may be used to store the recognized MIME types for a particular Plug-in. Microsoft Internet Explorer does not have this restriction with their extensible browser Objects architecture. Working within this limit, Oracle JInitiator will provide backward support for as many earlier MIME types as is possible.

The accompanying documentation and release notes for an Oracle JInitiator release will provide an accurate description of what MIME types are supported for that specific release.

Is it possible to make Forms Developer applications run in any version of Oracle JInitiator?

Yes. Oracle provides a generic MIME type that will allow any installed version of Oracle JInitiator to run the Forms Developer Application. This MIME type application, x-jinit-applet, is recognized by every version of Oracle JInitiator. Always using this MIME type will enforce the upgrading of later Oracle JInitiator versions by the browser.

B.3.4 Operation of Oracle JInitiator

Can the Forms Applet window be run within the same browser window from which it was launched?

Forms Server Release 6*i* supports the running of the Forms applet both within the same browser window and in a new window. This is a configurable option and is set as a parameter in the base HTML file.

What happens to the running Forms Developer application if the user navigates off of the current browser page?

Oracle JInitiator contains an additional feature that allows a running Java application to be cached and retrieved when required during the current browser session. This means that when a Forms application is run and the user navigates to a different page and then comes back to the Forms application page, the running Forms application will appear exactly as it was when the user left it.

Can I use the Oracle JInitiator to run my custom developed Java applications?

Oracle JInitiator uses a standard JavaSoft JVM that has been enhanced by the Oracle development team. It should be capable of running custom Java applications. However at this time, Oracle only provides support for Oracle JInitiator when running Oracle Java-based applications, such as Forms Developer, Oracle Enterprise Manager, and Oracle Discoverer. The use of Oracle JInitiator to run custom Java applications is not supported by Oracle.

Can Oracle JInitiator and the JavaSoft Java Plug-in coexist on the same machine?

Yes. They can coexist in the same browser installation because they use different MIME types to launch the plug-in.

Will Oracle JInitiator coexist and operate correctly when used at the same time as the Javasoft Plug-in, in the same browser instance?

No. Due to the way that dynamically loadable libraries are loaded and the JVM dynamically loadable libraries are named, the Oracle JRE and the JavaSoft JRE can

not be run simultaneously from within the same browser instance. This means that a browser user cannot switch from using the JavaSoft Java Plug-in to Oracle JInitiator in the same browser instance. The browser must be stopped and restarted when switching between the different applications that use Oracle JInitiator and Java Plug-in from JavaSoft.

With the JavaSoft Java Plug-in and Oracle JInitiator there is an option to use a different JRE. Can I use the JavaSoft Java Plug-in when it is configured to use the Oracle certified JRE to run Forms Developer applications?

The only certified and supported combination is Oracle JInitiator with Oracle JRE. The Oracle JRE, while conforming to the JavaSoft standard, contains bug fixes to the JavaSoft JRE that allow Forms Developer applications to run correctly. Oracle works closely with Javasoft to ensure that Oracle's enhancements are communicated to JavaSoft and applied to the standard JRE, but is unable to wait for the improved JavaSoft JRE to be released.

The figure below shows the Oracle JInitiator Control Panel and the correct settings for the Java Run Time Environment value.

B.3.5 Caching

Can Oracle JInitiator cache the Java class files downloaded when an application is run? If so, does this mean the Java class files are downloaded only once and not each time the application is started?

Yes. Oracle JInitiator provides a persistent caching mechanism for JAR files that it downloads when running Java applications. A JAR file is a standard Java archive that contains a series of Java class files that are used by the Java application. By putting all the required class files into a single JAR file, a single download is performed rather than multiple downloads for each individual class file required.

By caching the JAR files on the client, Oracle JInitiator alleviates the need to download the JAR files each time they are required for an application. The first time a JAR file is required it is downloaded from the Web server and then saved to the local client machine. The next time it is required, Oracle JInitiator will look into the cache directory to see if the file is stored there; if it is, it will use it from the local directory and avoid having to re-download the file from the Web server. This saves a lot of user time and network traffic for commonly used applications. For example, if your application uses a 2MB JAR file and you have a fast Ethernet connection that is capable of downloading a 2MB file in 5 seconds then you will save 5 seconds at application startup. If you are running on a slow dial-up network that takes 10 minutes to download a 2MB file, then you will save 10 minutes at application startup.

How does Oracle JInitiator caching technology work?

Oracle JInitiator provides browser-session-independent caching of JAR files. Oracle JInitiator stores the downloaded JAR files on the local client machine so that it does not need to download them the next time they are required.

When a JAR file is requested, Oracle JInitiator will check the cache directory to determine if the file has been previously requested, downloaded, and stored. If the JAR file is not present, Oracle JInitiator will download the JAR file from the Web server and then store it for future use in the cache. Some additional information is stored in the cache file to enable Oracle JInitiator to uniquely identify the JAR file as well as the Last-Modified date of the requested file as reported by the Web server.

If the file is present in the cache, then the Web server must be checked to determine if the stored JAR file is current. Oracle JInitiator takes the Last-Modified date contained in the cached JAR file and asks the Web server (using standard HTTP interactions) if the file on the server has been modified. The Web server uses the given Last-Modified date and the timestamp on the file stored on the server. Then it either serves the newer file to Oracle JInitiator with a status code of 200 or returns a status code of 304, which indicates that the file in the cache is current.

If the cached JAR file is not current, a new one is downloaded and stored for future use in the cache directory. If the file is current, Oracle JInitiator loads it from the cache directory and updates the timestamp on the cached file to indicate the last time it was used.

Where do the cached JAR files get stored?

By default, Oracle JInitiator stores the downloaded JAR files in the jcache subdirectory, which is located in the Oracle JInitiator installation directory.

Why does the jcache directory contain strange names for the cached JAR files?

Since each JAR on a Web server can be identified by a URL (URL = codebase + JAR filename), the Oracle JInitiator caching mechanism uses this to uniquely identify the JAR file. On Windows operating systems, since the full URL is not a valid filename for a file, Oracle JInitiator transforms it via a simple hashing algorithm into an acceptable filename and then uses this as the stored JAR filename. When a request is made for a JAR file, Oracle JInitiator performs the hashing algorithm on the complete URL and then checks to see if the resulting filename exists in the cache.

How does JAR file caching work with server load balancing?

As outlined previously, JAR files are identified in the cache based on the URL from which they were retrieved. Consequently, the same JAR file from different servers will be downloaded from each different server. This is done deliberately to ensure

security and application integrity. If JAR files were cached solely using their name, then a malicious application could replace the JAR file from another application. When the original application was run, the Java class files would be different. Also, since JAR files are not guaranteed to have unique names, it is possible for JAR files to collide. This would happen where two different applications use the same JAR filename, but require different class files from the JAR file.

It appears that the timestamp on the cached JAR files is updated every time I run an Forms Developer application. Is this normal? Does it mean that the file is being downloaded every time?

No. Oracle JInitiator supports a configurable cache maximum size. Every time a cached JAR file is used, Oracle JInitiator updates the timestamp to indicate the date and time that the cached file was last used.

If the cache size grows to the point where files must be removed in order to maintain the maximum cache size, Oracle JInitiator uses the timestamp of the cache files to determine which is the least recently used file and then removes that.

How can I tell that my cache is functioning correctly and that the JAR files are not being downloaded every time?

When Oracle JInitiator needs to download a required file, it does so via the Web server that has been configured to run Forms Developer applications. Modern Web servers support the use of log files that enable the tracking of what files have been downloaded, by whom, and when. The Web server log file uses a standard format to describe the transactions that have occurred. This log format includes the name of the requested item and the result of the request. The result of the request is indicated using a set of standard HTTP status codes.

If the JAR file was downloaded to the client, the log file will contain the name of the requested JAR file and the HTTP status code 200. If the JAR file was not downloaded because the timestamp on it was earlier than the cached file timestamp, then the log file will contain the name of the requested JAR file and the HTTP status code 304.

The following example shows an entry made in a log file using standard NCSA log formatting when the JAR file in the cache is not current and must be downloaded from the Web server.

```
ferret.us.oracle.com - - [19/Feb/1999:17:40:12 -0800] "GET
/forms_java/f60all.jar  HTTP/1.0" 200 -
```

The following example shows an entry made in a log file using standard NCSA log formatting when the JAR file in the cache is current and is therefore not downloaded from the Web server.

```
ferret.us.oracle.com - - [19/Feb/1999:17:42:29 -0800] "GET
/forms_java/f60all.jar  HTTP/1.0" 304 -
```

It seems that when the JAR file is downloaded, a .JCX file is created in the jcache directory. What is this file?

As the JAR file is being downloaded a temporary copy of it is written to the file system. This temporary copy is identified by the .jcx file extension. Once the download has successfully completed, the .jcx file is moved to a .jc file. If the download is interrupted at any point or the connection is dropped, the operation will not be complete and the temporary file will remain with a .jcx extension. Oracle JInitiator will not load a file with a .jcx extension since it is not valid.

I've verified that the caching is working correctly, but my application is still taking longer to start than I'd like. Why is that?

The JAR file caching provided by Oracle JInitiator does not perform any magic to increase the speed of Java on your system. What it does is save you the time it requires to download the required JAR files for each application startup. The operation of unzipping a JAR file, loading the contained classes into memory, and then authenticating them to ensure that they have not been tampered with takes a significant amount of the startup time. In fact, on a very fast network the amount of time taken to download the JAR file will be smaller than the amount of time required to load the Java classes into memory and perform the authentication. This means that caching saves you very little in terms of overall application startup. On a slower network, the time required to download JAR files will become proportionately larger in the overall startup time, so JAR file caching becomes more important.

C

AppletViewer

C.1 Introduction

This appendix describes the AppletViewer, an alternative to using Oracle JInitiator. The AppletViewer is a JDK component and an Oracle-supported product that client machines use to view applications running on the Forms Server. Upgraded versions are available for download from the Forms Developer Web site.

Note: The AppletViewer is only supported on Windows 95 and Windows NT 4.0.

C.2 Running Application in the AppletViewer

To run applications in the AppletViewer, you must complete the following steps:

- Prepare to run your application with the AppletViewer.
- Add the clientBrowser parameter to your base HTML file.
- Set the clientBrowser parameter.

When running your application in the AppletViewer, requests to show a URL (for example, web.showDocument and RUN_PRODUCT) will be ignored by the AppletViewer. If this is the case, you will need to follow the process to trust the Forms applet, as described later in this chapter in Section C.3.1, "Trusting the Forms Applet by Registering Its Signature".

C.2.1 Preparing to Run Your Application with the AppletViewer

In order to prepare to run your application within the AppletViewer, make the AppletViewer available for download and inform your users that they will have to install the AppletViewer on their client machines. Complete the following:

1. Customize JDK_DOWNLOAD.HTM.

   JDK_DOWNLOAD.HTM is the template HTML file that allows your users to download the AppletViewer.

2. Copy JDK.EXE to your Web server.

   You must copy JDK.EXE to the location specified within JDK_DOWNLOAD.HTM.

3. Copy JDK_DOWNLOAD.HTM to your Web server.

   You must copy JDK_DOWNLOAD.HTM to the location specified within JDK_DOWNLOAD.HTM.

C.2.2 Adding the clientBrowser Parameter to your Base HTML File

To use the clientBrowser parameter, you must have security permissions to issue a system call that executes the named application. In general, when loading Java class files, the Forms applet is not trusted and, as such, cannot issue such system calls. However, when the Forms applet is trusted, it is able to issue these calls. The Forms applet is considered trusted when one of the following is true:

- The Forms applet signature is "registered" on the client machine as described in Section C.3.1, "Trusting the Forms Applet by Registering Its Signature".
- The Forms Java class files are installed locally on the client system and the CLASSPATH environment variable is set as described in Section C.3.2, "Trusting the Forms Applet by Installing the Forms Java Class Files Locally".

These HTML file examples assume that you trusted the Forms applet by registering its signature on your machine. If you trusted the Forms applet by locally installing the Forms Java class files instead, you should not download the F60ALL.JAR file. Therefore, remove the ARCHIVE="/.../f60all.jar" applet tag from your HTML file.

C.2.3 Setting the clientBrowser Parameter

To set the clientBrowser parameter, do one of the following:

- Add the clientBrowser parameter to your HTML file.
- Add the clientBrowser parameter to your HTML file, and have each client modify their JDK_SETUP.BAT file.

Add the clientBrowser Parameter to Your HTML File.

This option assumes that every client has its browser executable installed into the same physical directory because the physical path of the browser is hard-coded in the HTML file. For example:

```
<APPLET CODEBASE="/forms60code/"
     CODE="oracle.forms.engine.Main"
     ARCHIVE="/forms60code/f60all.jar"
     HEIGHT=480
     WIDTH=640>
     <PARAM NAME="serverArgs" VALUE="module=start.fmx userid=scott/tiger">
     <PARAM NAME="clientBrowser"
     VALUE="c:\programfiles\netscape\communicator\program\netscape.exe">
</APPLET>
```

Add the clientBrowser Parameter to Your HTML File and Have Each Client Modify Their JDK_SETUP.BAT File.

This option is best if there is a possibility that clients have installed their browser executables into different physical directories. It does assume, however, that all clients are using the same browser. For example, the HTML file might look like this:

```
<APPLET CODEBASE="/forms60code/"
     CODE="oracle.forms.engine.Main"
     ARCHIVE="/forms60code/f60all.jar"
     HEIGHT=480
     WIDTH=640>
     <PARAM NAME="serverArgs" VALUE="module=start.fmx userid=scott/tiger">
     <PARAM NAME="clientBrowser" VALUE="netscape">
</APPLET>
```

And JDK_SETUP.BAT would look like this:

```
SET CLASSPATH=C:\ORANT\JDK1.1\JDK\LIB\CLASSES.ZIP
PATH C:\PROGRAM FILES\NETSCAPE\COMMUNICATOR\PROGRAM;
C:\ORANT\JDK1.1\JDK\BIN;%PATH%
```

C.3 Registering the Forms Applet Signature

A signature allows client machines to verify that a file has been downloaded from a valid and trusted entity (a *signer*). This allows client machines to protect themselves from malicious or malfunctioning Java archive (JAR) files. In order for a JAR file to be validated by a client, the signature of that file must be registered on the client machine. Javakey is a Sun Microsystems command-line tool that generates digital signatures for JAR files.

The Forms applet is itself a signed JAR file. You have two options for registering the Forms applet signature. Choose one of the following:

- Register the signature on your client machine(s) using the Forms applet signature we provide.
- Re-sign the Forms applet with your own signature and register that signature on your client machine(s). If you choose this method, please refer to http://java.sun.com/security/usingJavakey.html for instructions on creating and signing JAR files.

C.3.1 Trusting the Forms Applet by Registering Its Signature

To trust the Forms applet by registering its signature:

1. Copy the Forms Developer certificate to \ORACLE_HOME\FORMS60\JAVA on the client machine.

   The certificate is a file named Dev.x509. It is located in \ORACLE_HOME\FORMS60\JAVA on the server.

2. Open a DOS Command Prompt, and navigate to \ORACLE_HOME\FORMS60\JAVA.
3. Type: javakey -c Developer true

   This command creates a trusted identity for the AppletViewer on the client's identity database using the exact name of the certificate provider.

4. Press Enter.
5. Type javakey -ic Developer Dev.x509

   This command imports the Dev.x509 certificate into the client's JDK identity database and associates the certificate with the trusted identity created in step 3.

6. Press Enter.

C.3.2 Trusting the Forms Applet by Installing the Forms Java Class Files Locally

To trust the Forms applet by installing the Java class files locally:

1. Copy the \ORACLE_HOME\FORMS60\JAVA directory to a new directory on the client machine.

   Copy this directory exactly; do not change the directory structure in any way.

2. Modify JDK_SETUP.BAT in your ORACLE_HOME directory:
   a. Open JDK_SETUP.BAT in a text editor.
   b. Modify the CLASSPATH environment variable to reference the new directory.
   c. Save your changes to JDK_SETUP.BAT.

C.4 Instructions for the User

To run an application from within the AppletViewer, complete the following steps:

- Install the AppletViewer.
- Run the AppletViewer.
- Invoke a Web browser from within the AppletViewer.

C.4.1 Installing the AppletViewer

To install the AppletViewer, use the Oracle Installer to install the JDK AppletViewer:

1. Shut down any active Windows applications.
2. From the taskbar, choose **Start → Run**.
3. In the Run dialog, type the following (where D: is your CD-ROM drive letter): `D:\setup.exe` and click **OK**.
4. In the Oracle Installation Settings dialog, check the default values for your company name and your ORACLE_HOME directory.
5. Click **Oracle Forms Server**.

6. Click **Custom**.
7. From the list of Available Products, select **JDK AppletViewer**.
8. Click **Install**.

C.4.2 Running the AppletViewer

To run the AppletViewer:

1. From a DOS command, navigate to the AppletViewer executable (appletviewer.exe).
2. Run the AppletViewer executable, specifying the host name, HTML file virtual directory, and HTML file.

   For example, type: appletviewer http://myhost.com/web_html/start.html

3. Press Enter.

C.4.3 Invoking a Web Browser From Within the AppletViewer

To invoke a Web browser from within the AppletViewer:

1. Trust the Forms using one of two methods:
   - Register the Forms applet signature.
   - Install the Forms Java class files locally.
2. Add the clientBrowser parameter to your base HTML file.

Part III

Index

Index

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W