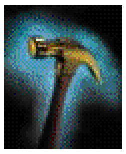


An Overview of Oracle® Forms Server Architecture

An Oracle® Technical White Paper
April 2000

INTRODUCTION

This paper is designed to provide you with an overview of some of the key points of the Oracle® Forms Server architecture and the processes involved when forms are deployed over the Web.

Using the Forms Server you can run complex applications over the Internet, without compromising either functionality or richness of interface. You can build new applications specifically for Web deployment, or take your existing Forms, Menus and Libraries that are currently deployed in Client Server and move them to Web deployment almost without change. There are some restrictions inherent in using Forms over the web, and these are explained in the section "Forms Server Restrictions."

The architecture of the Forms Server is more complex than that of the conventional Client/Server implementation of the product, so let's examine it in detail.

Components

The Forms Server consists of a Java Client that is downloaded automatically to the end user and three components in the middle tier "Application Server:" Figure 1 illustrates how a form runs on the web.

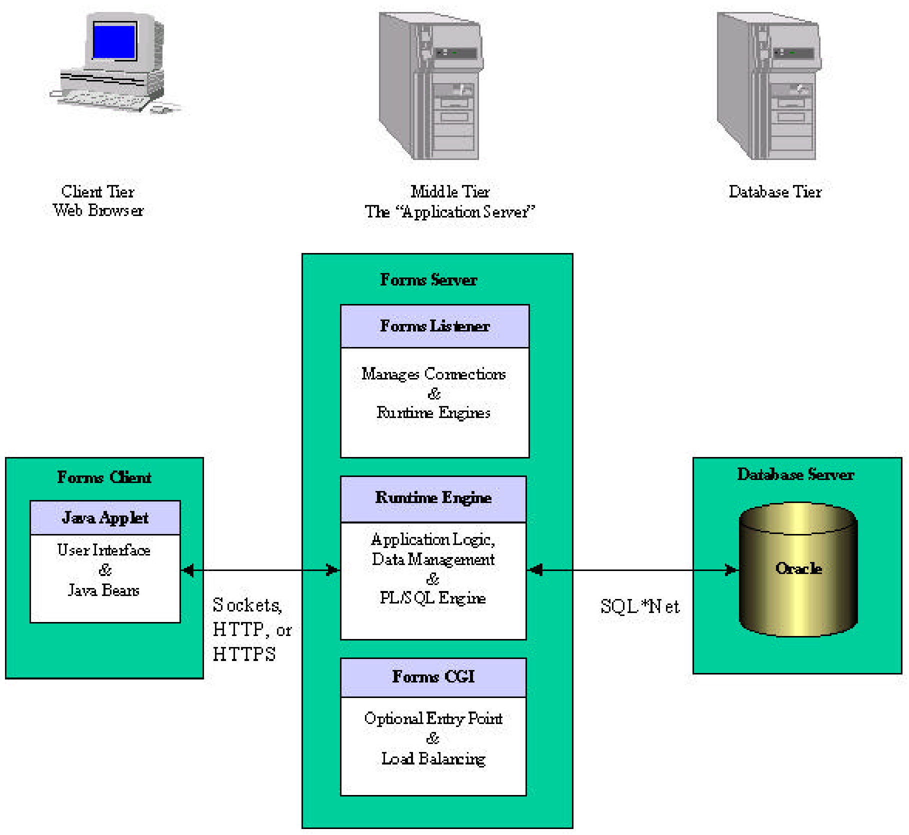


Figure 1. Running a Form on the Web

- ***Java Applet***: When a user runs a Forms session over the web, a thin 100 percent pure Java Client dynamically downloads from the Application Server. This Java Client provides the user interface for the associated Forms Server Runtime Engine on the server, and handles user interaction and visual feedback such as that generated by navigating between items or checking a checkbox. This applet is the same for any Form that is run using it, you don t have to generate Java code for every application or form that you want to web deploy.

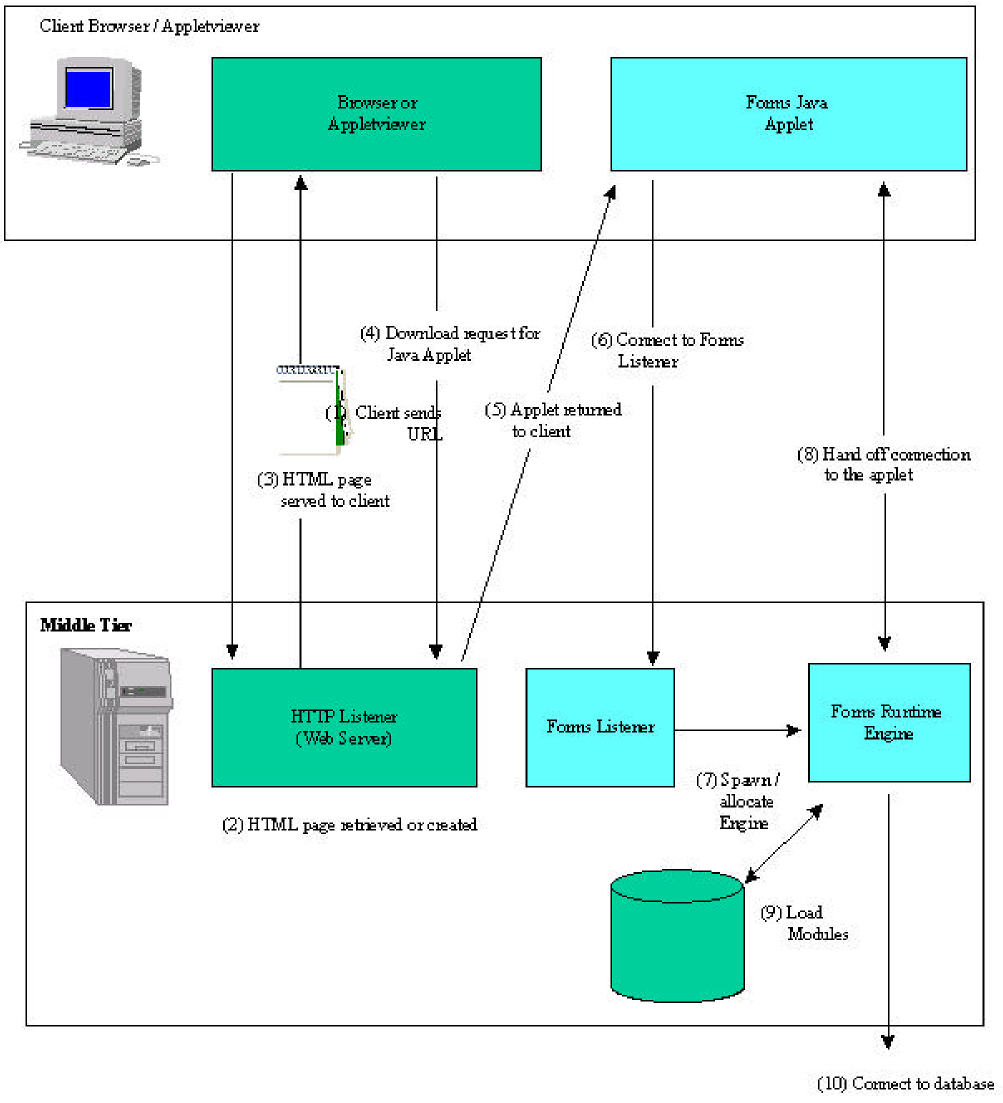
- ***Forms Server Runtime Engine***: The Forms Server Runtime Engine performs the same function as the Client Server Runtime Engine, except that all user interface functionality is redirected to the Java Client (above). The Forms Server Runtime Engine is the process that maintains a connection to the database on behalf of the Java Client. The code that will be run by the Forms Server Runtime Engine are the same Forms, Menus and Libraries files that would be used for running in Client/Server mode on the same platform. No conversion or re-compilation is required to web-deploy an application!

- ***Forms Server Listener***: The Forms Server Listener acts as a broker, taking connection requests from the Java Client processes and initiating a Forms Server Runtime process on their behalf. The Listener can also maintain a pool of running engines ready for connection, making the connection from the Java Client complete as quickly as possible.

- ***Forms CGI*** *(optional)*: The Forms GCI provides a single entry point to many different Forms applications and administers load balancing.

CONNECTION PROCESS IN DETAIL

We've looked at the components that make up the Forms Server, now let's look at the connection process itself. Figure 2 illustrates the steps that establish a Forms session with the Forms Server.



Client Browser / Appletviewer
Browser or Appletviewer
Forms Java Applet
(4) Download request for Java Applet
(6) Connect to Forms Listener
(1) Client sends URL
(5) Applet returned to client
(8) Hand off connection to the applet
(3) HTML page served to client
Middle Tier
HTTP Listener (Web Server)
Forms Listener
Forms Runtime Engine
(7) Spawn / allocate Engine
(2) HTML page retrieved or created
(9) Load Modules
(10) Connect to database

Figure 2. Forms Connection Process in Detail

1. The user chooses a Link from a web page, types a URL directly or passes the URL as an argument to the applet viewer.

2. The Web Server interprets the URL that is passed and displays an HTML page containing an <APPLET> tag that describes the Forms Java Client to the Browser. The HTML page may be a *static* HTML page that has all of its parameters hard-coded into it , or the URL that is passed could call the Forms Server CGI program to create a HTML page dynamically.

3. The Client receives the HTML file served by the Web Server. The <APPLET> tag in the HTML file will supply the information required to locate the Java Class files that make up the Forms Java Client. Within the APPLET tag in the HTML file you would also supply information about the Form that should be run, and any other parameters that you want to pass to your Forms session, such as the Login information. The APPLET definition also contains instructions on what Forms Server to run and many parameters which can help you to customize aspects of the Java Client such as the Look and Feel, color schemes etc. The HTML file might also contain other HTML tags such as those to tell the browser to run this particular applet using the JInitiator Plug-in (see JInitiator).

4. The Browser asks the Web Server for the Java Class files from the location specified in the HTML file. The APPLET CODEBASE parameter in the HTML file is used to define this. The files may be downloaded individually or as an "Archive". This archive will have an extension of .JAR and can be best thought of as a .ZIP file containing all of individual .CLASS files required by the Applet. The use of a JAR file speeds up the download of the Java Client. The ARCHIVE parameter defines which (if any) .JAR file should be used. If this particular HTML file specified that JInitiator should be used, then JInitiator will carry out the additional step of checking the version of the Forms Client Java code available on the Web Server and will only download it if it turns out to be newer that any version that JInitiator currently has cached.

5. The .CLASS or .JAR files are downloaded (if not already present) to the Browser and the Java applet starts.

6. The Java Client applet connects to the Forms Server Listener by talking to the TCP/IP socket or HTTP port defined by the *serverPort* parameter in the APPLET Tag. The Forms Listener will have already been started and will be listening for incoming requests on this particular socket.

7. After receiving the connection request from the Java Client, the Forms Listener starts a new Forms Runtime engine for this client. You can configure the Forms Server to hold a pool of unconnected Runtime engines which can be allocated as connection requests come in, thus speeding up the whole connection process.

8. The Forms Server Listener hands off the connection to the Java Client and has no further part in the process.

9. The Forms Server Runtime Engine allocated to this client takes over, loading the module specified in the serverArgs entry in the HTML file and any libraries and menus that are required by that form.

10. The user is prompted for database login information, if this had not already been supplied, and the connection to the database server is established. The user is now ready to work.

FORMS SERVER LOAD BALANCING

The Forms Developer Server provides the facility to load balance, when using the Forms CGI program. This allows you to maintain a pool of middle tier machines a "Server Farm," each of which would run a Web Server and the Forms Server.

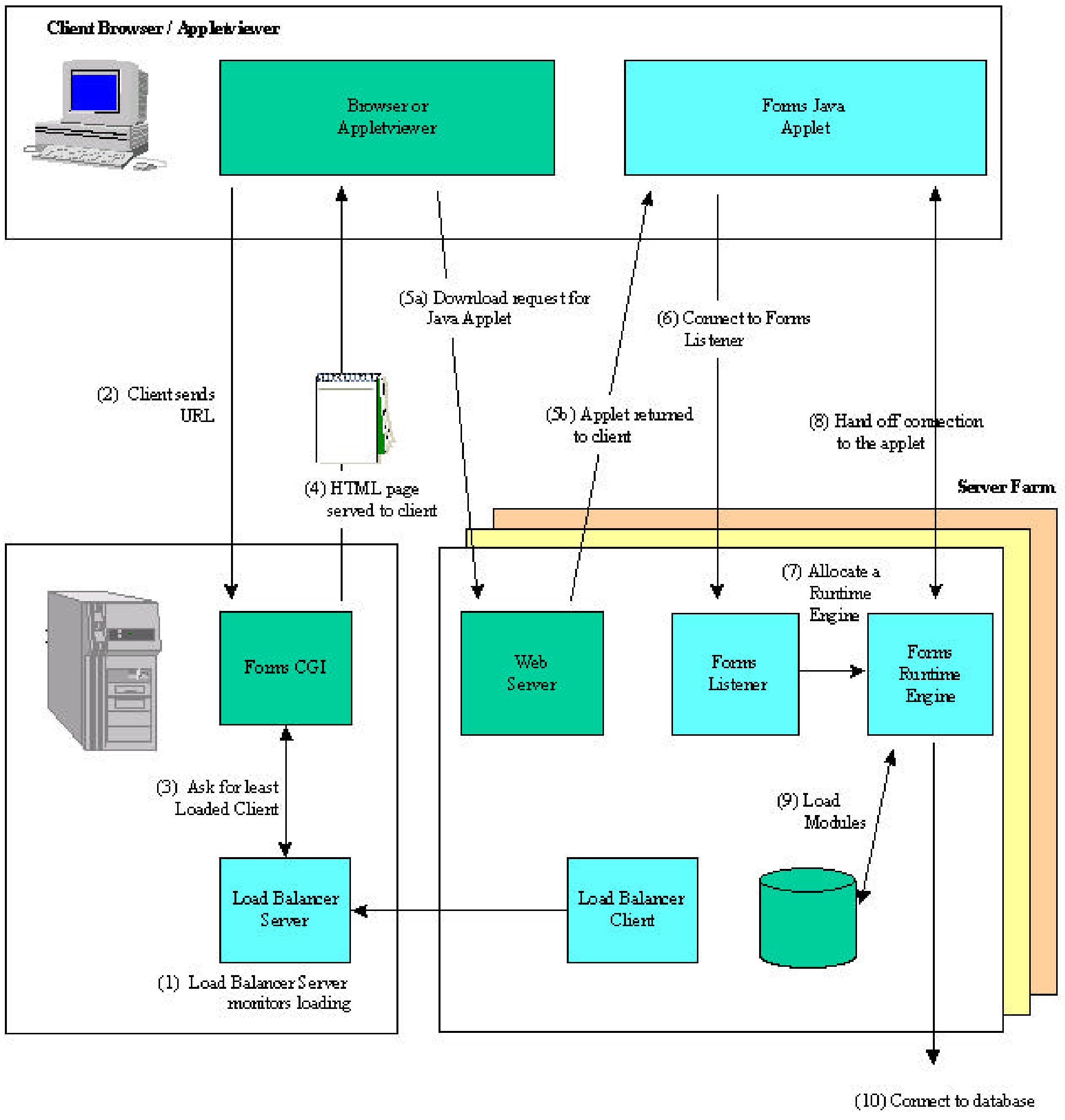
Load Balancing provides you with a more scaleable configuration. When you approach the limits of your current hardware, rather than either upgrading or throwing out that machine, you can just add more nodes to run your application and spread the increasing load across several machines rather than one.

When users connect to the Forms CGI (see below) that is set up for load balancing, two extra components of the Forms Server are utilized:

- **Forms Load Balancing Server** The Server maintains information about all machines currently allocated to the "Server Farm" and their current loading.

- **Forms Load Balancing Client** The Client runs on each of the machines allocated to be in the "Server Farm," its job is to simply report to the Load Balancing Server with loading information, that is, the number of Forms processes that are currently running on that particular machine.

When a user connects to the Forms CGI in a load balancing situation, the sequence of events to create a Forms session is slightly different.



Client Browser / Appletviewer
Browser or Appletviewer
Forms Java Applet
(2) Client sends URL
(4) HTML page served to client
(5a) Download request for Java Applet
(5b) Applet returned to client
(6) Connect to Forms Listener
(8) Hand off connection to the applet
Server Farm
(7) Allocate a Runtime Engine
Forms CGI
Web Server
Forms Listener
Forms Runtime Engine
(3) Ask for least Loaded Client
(9) Load Modules
Load Balancer Server
Load Balancer Client
(1) Load Balancer Server monitors loading
(10) Connect to database

Figure 3. Running a Form in a Load Balancing Scenario

1. The Load Balancing Server is continually appraised of the process loading on each of the Load Balancing Clients on machines within the server farm.

2. In this case, the URL supplied by the user is passed to the Forms CGI.

3. The Forms CGI will already have been configured to know that it should be load balancing and will ask the Load Balancing Server for the member if the Server Farm that currently has the lowest number of Forms Server processes running. If your Server Farm consists of a mix of different hardware platforms or configurations then scaling factors can be applied when starting the Load Balancing Clients. This scaling factor will allow you to skew the loading in favor of more powerful machines, rather than the choice of least loaded server being determined solely by the number of processes running.

4. Having determined the machine in the Server Farm that is best able to handle this new connection, the Forms CGI will build an HTML file pointing the browser at that server for the download of the Java Applet and connection to a Forms Server Listener.

5. The user's browser will now download the Forms Java Applet from this dynamically determined location, and thereafter the rest of the connection process will take place to this machine.

COMMUNICATIONS AND SECURITY

When running the Forms Server, the Java Applet implementing the user interface and the Forms Server engine need to communicate. This can be done using a conventional TCP/IP Sockets connection for intranet use, or using HTTP/1.1 as a protocol for Internet use where Firewalls prevent the use of a direct socket connection.

In either case, all of the traffic that is transmitted between the Client and Forms Engine is encrypted using 40-bit RSA encoding. This encryption can be turned off optionally, but we would not recommend that sites do this.

Also available as an option is the use of Secure Sockets Layer (HTTPS) instead of Sockets or HTTP to provide a further layer of security. HTTPS currently supports 128-bit encryption for US domestic customers and 40-bit for non-US customers. As the US relaxes laws for the export of encryption technologies, 128-bit encryption will become available for all users.

For more information on the use of HTTP with Oracle Developer Forms Server and security/firewall issues generally see the Oracle White Paper: *Deploying Internet Applications using HTTP Enabled Oracle Developer Server*.

SECURITY CONSTRAINTS IMPOSED BY JAVA

As the thin client portion of the Forms Server is written in Java, it has to comply with certain restrictions that Java Applets[1] have imposed upon them. The first thing that you would notice when running a Forms application over the Web is that the Windows have a bright yellow bar across the button bearing the legend "Warning Applet Window". This is Java's way of reminding you that you downloaded this program from someone you can't necessarily trust!

[1] It is important to note the difference between a Java Applet and a Java application. As an Applet is by definition downloaded off of the Web, the Java virtual machine that runs in the web browser or appletviewer strictly controls what the applet can do. This is termed the sandbox; applications running within the sandbox do not have access to objects outside of it such as memory, the local hard drive, printers etc. So if you access a web page that attempts to download a rogue Java application onto your machine, it won't be able to escape from the sandbox and therefore won't be able to damage your machine.
In the case of a Java Application, this is where the Java code is sourced from your machine and therefore is implicitly trusted to behave responsibly with your hard-disk etc. in the same way that any normal 'C' program would be.

As well as the warning you'll find that the Java Virtual Machine prevents you from carrying out simple tasks such as printing the screen or copying data onto the clipboard.

These constraints can of course be a little too restrictive, when you do in fact, know where the code has come from, which is true in the case of Oracle Forms Developer. Therefore you can choose to "trust" Oracle Forms Developer on your machine, by registering a Certificate that we supply with the Java Virtual Machine. Then, when you download an applet which claims to be the Forms Server Applet, the JVM will check its credentials using the stored certificate information, and if all matches, the applet will be trusted and able to print, use the clipboard and of course you don't get the bright yellow warning any more!

When you install Jinitiator, we automatically register the Forms Certificates for you.

PERFORMANCE AND SCALABILITY

A lot of work has been done with the Forms Developer Server to make it as performant as possible from end to end and giving you performance that you would expect from a Client-Server implementation, but with the vastly reduced administrative overheads of Web Deployment, and no client software installs!

Details of how to leverage the maximum scalability and performance from the Forms Server can be found in the White Paper: *Oracle Developer Server: How to Optimize the Deployment of Internet Applications.*

Additionally, if you would like more "Real" World" information about Oracle Developer Server scalability and performance refer to the following White Papers:

- Scaleable Web Deployment with Oracle Developer Server – A Benchmark Comparison of Client/Server and Web Deployment by Retek Information Systems (December 1998).
- Developer Server Scalability Testing (December 1998).

JINITIATOR

JInitiator is Oracle's version of Sun's Java Plug-In, which provides the ability to specify the use of a specific Java Virtual Machine on the client instead of using the browser's default JVM. Oracle JInitiator runs as a plug-in for Netscape Navigator and as an ActiveX component for Internet Explorer; allowing customers to run Oracle Developer Server applications using Netscape Navigator or Internet Explorer.

The Oracle JInitiator provides these major benefits:

- Allows the latest Oracle-certified JVM to run in older browser releases.
- Ensures a consistent JVM between different browsers.

- Provides functional extensions to the basic JVM such as SSL support.
- Provides a reliable deployment platform. Oracle JInitiator has been thoroughly tested and certified for use with the Oracle Developer Server.
- Is a performant deployment environment. Application class files are automatically cached by the JInitiator, providing fast application start-up.
- Is a self-installing and self-updating deployment environment. Oracle JInitiator automatically installs and updates itself like a plug-in or an Active-X component. Local cached application class files will be automatically updated from the application server based on a date-time stamp comparison.

FORMS SERVER RESTRICTIONS

Although the Forms server allows you to deploy your existing applications without major changes, some restrictions are imposed by the three-tier architecture which you will need to take into account:

- OCX, and VBX controls are not supported. As the user interface is not on the same machine as the runtime engine, there is no way for the control to display. As version 6.0 of Developer Forms Server supports the use of Java Beans, you can replace the functionality provided by the Windows Specific control with a similar Bean.
- User_Exits, ORA_FFI and Host commands all execute on the middle tier, not on the Java Client. This implies that they cannot carry out any user interface interaction as the display being used by the user is not the same display as the middle tier.
- TEXT_IO. Reading and writing files using the Text_io package will again take place in the middle tier. You cannot read a "Local" file off of the end users machine, unless the user's drive is shared and visible from the Forms Server machine, or alternatively you use a Java Bean.
- Mouse Move, Mouse Enter and Mouse Leave events. The Triggers that would normally be executed by Forms in a client server deployment have been disabled when running on the web. This prevents the network from being swamped with mouse messages between the Forms Java Client and the middle tier.
- Local devices. Customers often ask if they can utilise devices such as Barcode scanners attached to their browser machines and integrate them with Forms running over the web. By default this is not possible as there is (as was mentioned above) no way of programmatically accessing the browsers local machine from within the Java Applet. It is possible to implement this functionality by writing a trusted Java bean which can be embedded into the Forms application and access the local operating system.