How to Tune your Oracle Forms Server Applications

An Oracle Technical White Paper

March 2000

ORACLE

INTRODUCTION

This paper describes the tuning considerations when you deploy an application over the Internet or other network environment using the Forms component of Oracle Forms Server. This paper looks at the Java client, the network, and the resources on the application server. Tuning the connection between the Oracle Forms Server and the data server is beyond the scope of this white paper.

This paper describes the following:

- Deployment scenario for Oracle Forms Server.
- Optimizations that are *built into* the Oracle Forms Server platform.
- Effects of built-in optimizations "out-of-the-box."
- Examples of how to *tune* applications to fully exploit optimization features.

DEPLOYMENT SCENARIO

The following figure illustrates the fundamental deployment scenario of the Oracle Forms Server. The roles of each component are described in the sections that follow.

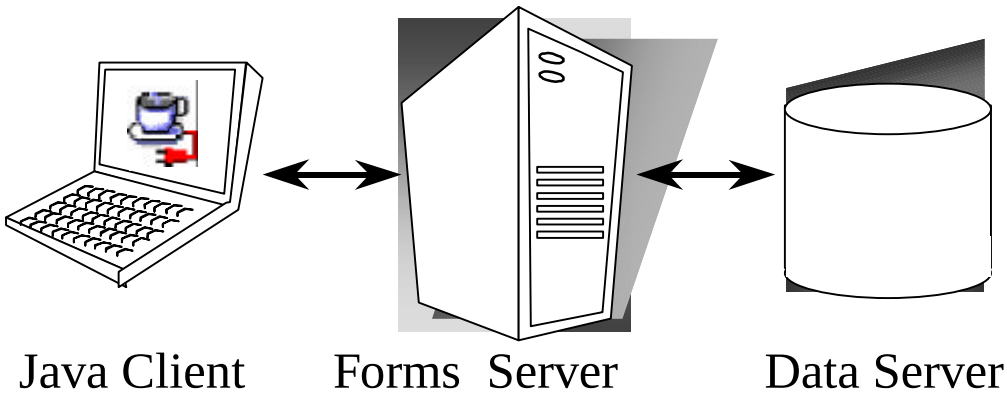

Figure 1: Deploying Forms with Oracle Forms Server

In Figure 1, the application is executed on the Oracle Forms Server while the application display is rendered by the Java client.

Typically, the data server and Oracle Forms Server are managed by a central IT group and benefit from a high-bandwidth, low-latency connection. Clients, however, are distributed throughout an organization and may connect to the Oracle Forms Server via several network hops. Therefore, clients will likely have higher latency and will have to share available bandwidth

with many other clients and applications.

ROLE OF THE ORACLE FORMS SERVER

The Oracle Forms Server is essentially a client to the data server. Therefore, all of the existing tuning techniques used for a client/server deployment can also be used between Oracle Forms Developer and the database when deploying Oracle Forms Server Internet applications.

Features such as server-side logic, use of Oracle8 OCI, array DML, and pre-fetch of data rows, are all excellent ways to optimize the connection between Oracle Forms Server and the database. Details on how to optimize the connection between Oracle Forms Server and the database are not within the scope of this white paper.

The Oracle Forms Server has additional functionality to efficiently communicate with a remote display on the Java client.

ROLE OF THE JAVA CLIENT

The Java client has three primary functions:

- Renders the application display for the user.
- Processes user interaction back to the Oracle Forms Server.
- Processes incoming messages from the Oracle Forms Server.

The communication protocols between the Java client and Oracle Forms Server are optimized for network bandwidth and latency. For example, the Java client processes messages from the Oracle Forms Server locally. At the same time, the Java client is common to all Oracle Forms Developer applications, so none of the application code resides locally.

BUILT-IN OPTIMIZATION FEATURES OF ORACLE FORMS SERVER

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

- Minimize client resource requirements.
- Minimize Oracle Forms Server resource requirements.
- Minimize network usage.
- Maximize the efficiency of packets sent over the network.
- Render application displays efficiently on the client.

These categories are described in more detail in the sections that follow. 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 Oracle Forms Server be minimized in order to contain the latency effects of the network.

MINIMIZE CLIENT RESOURCE REQUIREMENTS

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

The Java client is finely architected around many Java classes, which are grouped into functional subcomponents, such as displaying the splash screen, communicating with the network, and changing the look-and-feel. This granularity allows the developer and the Java Virtual Machine (JVM) to load functionality as it is needed, rather than downloading classes for functionality that is not required at the time.

MINIMIZE ORACLE FORMS SERVER RESOURCE REQUIREMENTS

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

Of the three sections shown, 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. Oracle 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.

MINIMIZE 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.

Oracle 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).

Oracle 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), Oracle 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.

MAXIMIZE 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 Oracle Forms Server.

The extensive use of triggers within the Oracle Forms Developer model is a strength, especially when used with 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 Oracle 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 Oracle 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 Oracle Forms Server as a single network message.

RENDER 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 Oracle 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.

EFFECTS OF BUILT-IN OPTIMIZATIONS

The following charts show the improvements gained by Oracle Forms Server "out-of-the-box" optimization features as

compared to Windows terminal and client/server implementations.

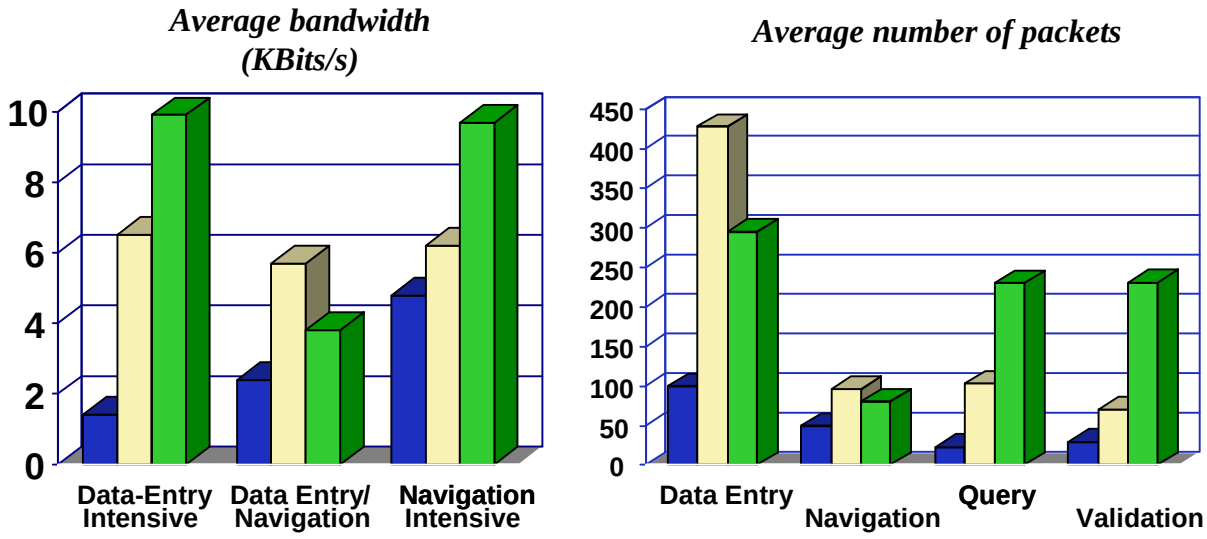

Figure 2: Improvements from Oracle Forms Server Optimization Features

Source: "Oracle Applications 10.7 NCA Scalability Benchmark" Technical White Paper

TUNING ORACLE FORMS SERVER APPLICATIONS FOR INCREASED PERFORMANCE

Oracle Forms Server automatically delivers the optimizations described in the previous sections. However, there are steps that an application developer can take to ensure that maximum benefits are gained from the built-in architectural optimizations. The remainder of this white paper discusses key performance issues that affect many applications and how developers can improve performance by tuning applications to exploit Oracle Forms Server optimizations.

Issues discussed are:

- Location of the Oracle Forms Server with respect to the data server.
- Minimizing the application start-up time.
- Reducing the required network bandwidth.
- Miscellaneous tuning techniques.

LOCATION OF THE ORACLE FORMS SERVER WITH RESPECT TO THE DATA SERVER

The Java client connection to the Oracle 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 Oracle 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 Oracle Forms Server in close proximity to the data server, which consequently locates the Oracle 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 Oracle Forms Server's improved efficiency over a network as compared to a traditional client/server implementation.

In the following example, the Oracle Forms 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) links.

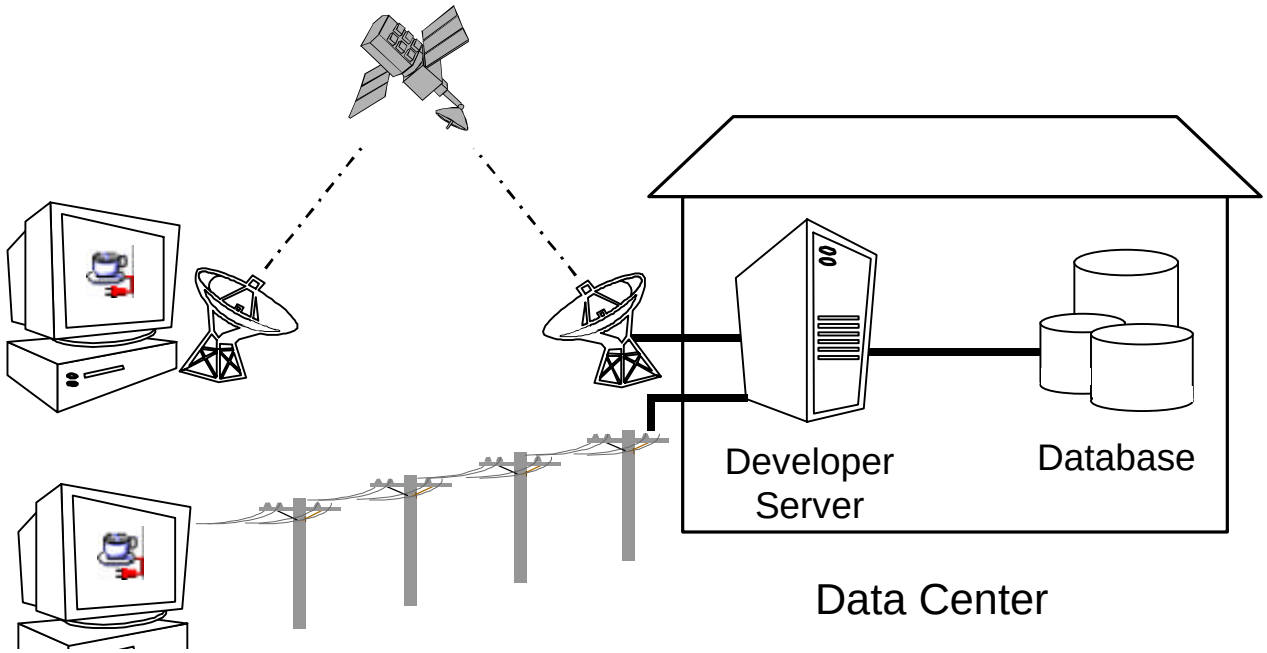

Figure 3: Co-Locating the Oracle Forms Server and Data Server

MINIMIZING THE APPLICATION START-UP TIME

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

minimize load time where ever possible.

After launching 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.
   - Authenticate security of classes.
3. Display splash screen.
4. Initialize Form.
   - Load additional Java classes, as required.
     - Authenticate security of classes.
   - Render boilerplate objects and images.
   - 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 Oracle Forms 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.
- Creating your own JAR files.

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.

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

File name	File size	Description
f60all.jar	1.57Mb	Contains all classes, and can be used to load all classes in bulk. Useful where network bandwidth is not constrained.
f60web.jar	1.4 Mb	Contains all classes except those used in LOVs. Use this in preference to f60all.jar when no LOVs are used in the application.
f60lov.jar	168 Kb	Classes specifically required to create/populate LOV dialog.
f60splash.jar	212 Kb	Useful for fast start up. Contains just enough classes to initialize the client and display the splash screen. Contains references to the remaining JAR files. (See "Deferred Load On Demand" later in this paper for details.)
f60help.jar	107 Kb	Classes specifically required to display help text.
f60common.jar	647Kb	Contains icons, windows handlers, common components, and network interfaces.
f60oracle_laf.jar	115 Kb	Contains Oracle look-and-feel icons and components.
f60resources.jar	20 Kb	Contains code to handle languages other than English.
f60rest.jar	519 Kb	Contains all other classes, such as support for HTTP communication and LOV.

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="f60web.jar,  icons.jar"

CODE="oracle.forms.."
```

Using Caching

Both of the supported JVMs for Oracle Forms Server (Oracle JInitiator and oJDK) 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 20 Mb. 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 (which is not described here) 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.

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 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.

The following example shows an application-specific JAR file that contains both a class required by the application and references to two independent Java Beans. When the JVM is required by the application to load Bean1, the initial JAR file points to the Bean1.jar file where all the Bean-specific classes and icons may be bulk loaded when required.

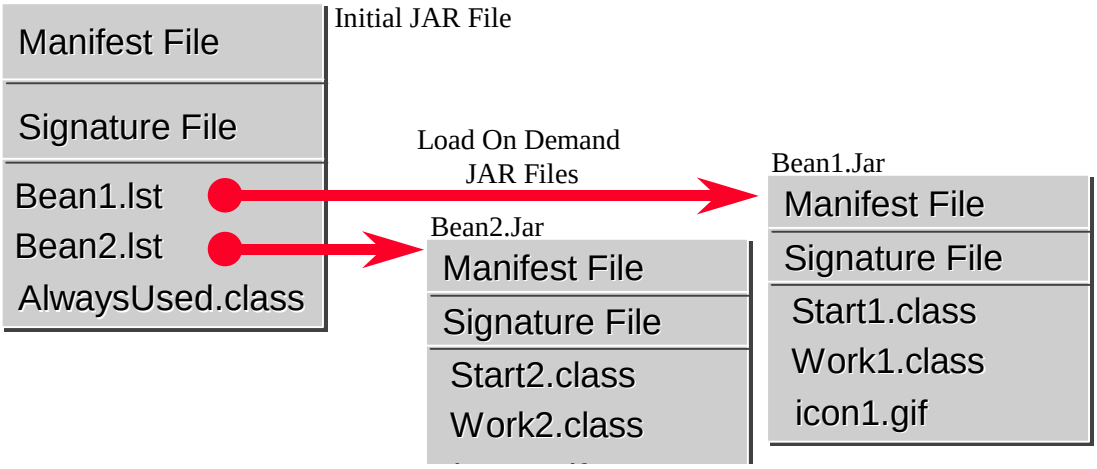

Figure 4: Using References to Other JAR Files

Creating Your Own JAR Files

Using the tools from Java Soft (contained in the oJDK), you can create or modify JAR files to tailor them for your application. The following are a few ideas for creating your own JAR files:

- **Store icons in the JAR file.** When an icon is referenced for a button, the client must download it from the server. This image is not cached, so redrawing a button or resetting an icon requires that the image be retrieved across the network again. By putting the icon in a JAR file, the image is cached by the JVM.

- **Store Java Bean classes in a JAR file.** If Java Beans are used in the application, group the classes into a single JAR file. If two Java Beans are not displayed at the same time, put them in separate JAR files to maintain granularity.

- **Create an application-specific manifest.** When an application has several JAR files associated with it (beyond those supplied by Oracle), create a JAR file that references the other JAR files to promote deferred loading-on-demand by the JVM (similar to the f60splash.jar example described earlier).

- **Sign only those classes requiring extra security.** When a class file is loaded, the JVM must authenticate any security associated with it, which can take a significant amount of time. However, secure classes are required so that the Java client can open a socket back to the Oracle Forms Server (on a machine other than the one from which it was loaded) or to hide the "Warning: Unsigned Applet" message on top-level windows. Oracle Forms Developer 6*i* and higher pre-signs only those classes that require extra security. You can create a custom JAR file that contains classes that require signing, which in turn requires that the developer obtain their own digital certificate. For more information on JAR security, see www.java.sun.com.

REDUCING THE REQUIRED NETWORK BANDWIDTH

The developer can design the application to maximize data stream compression by using "message diff-ing," described earlier in this white paper. The following steps can be taken to reduce the differences:

- Promote similarities between objects.
- Reduce the use of boilerplate text.
- Reduce the use of boilerplate items (such as arcs, circles, and polygons).
- Keep navigation to a minimum.
- Reduce the time to draw the initial screen.
- Disable MENU_BUFFERING.

These steps are described in more detail in the sections that follow.

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

Boilerplate text is a VGS object. It was commonly used in applications that were developed using previous versions of Oracle Forms Developer (before the PROMPT item property was available). The main disadvantage of boilerplate text is that it is passed as part of the VGS tree. Therefore, it will not benefit from "message diff-ing" or the string cache. As a developer, you should use the PROMPT item property rather than boilerplate text wherever applicable. Oracle Forms Developer 6.0 and beyond 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 (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 start-up times.

Reduce 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. For example, a wizard-type Form would usually include a "Next >>" button.

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 item properties:
  - RAISE ON ENTRY = YES (Canvas only)
  - VISIBLE = NO

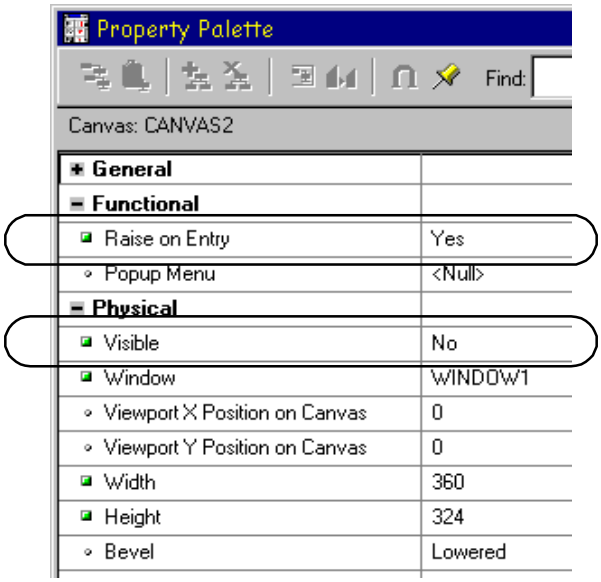

Figure 7: Hiding Elements that are not Immediately Required

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.

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 Oracle Forms 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 Java Beans.** 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 Java Beans that achieve the same effect without requiring the intervention of Oracle 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 Oracle 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 Oracle Forms 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.

- When using 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.

- When using 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 loaded again to the client, which requires network resources and start-up 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).

SUMMARY

Oracle Forms 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. In addition, Oracle Forms Server inherited all of Oracle Forms Developer's client/server features to ensure efficient communication with the data server.

As an application developer, you can further optimize performance by tuning your applications using the methods described in this white paper. You should fully explore the advanced optimization features available to tune an application before considering the load balancing features of Oracle Forms Server.

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
+1.650.506.7000
Fax +1.650.506.7200
http://www.oracle.com/