FORMS TUNING TECHNIQUES — USERS WILL SING YOUR PRAISE

Peter Koletzke, Quovera

Ever had this happen? You finish a project using Oracle Forms Developer. Everything has been tested and users seem to be happy, but a week later, you start getting calls that voice dissatisfaction on the speed of the application. Nothing like this happened while you were developing the form so you are at a loss. It is possible that you have a tuning problem. As with other types of tuning, application tuning is a critical activity. User perception of your application and even the application's success are affected by the responsiveness of the forms you create. The small effort you give to making your Forms applications perform efficiently will reap great benefits and have users singing your praise.

Application tuning is something that needs to be incorporated into the design and development of any system. If you properly tune your work before it is rolled out, you will not have to apply emergency measures to tune a production application. Of course, there are always performance issues that occur once an application has been in use for some time, but you can bypass some of these issues by writing efficient code from the start.

This paper explains techniques that you can use to make your forms run faster and more efficiently. It assumes that you have performed the usual steps of making the SQL and PL/SQL code the most efficient. There are many white papers that have been written about SQL and PL/SQL tuning, so the paper does not delve into that subject. It does provide an explanation of the areas you would look to for performance improvements outside the form and inside the form. It also discusses the special techniques that you can use to make your Forms code run more efficiently when deployed on the Web. Finally, the paper will mention the facilities offered by Oracle for benchmarking tuning and provide a brief bibliography of white papers and other sources that you can refer to for more information.

TUNING OBJECTIVES

You want your forms to be as efficient as possible, and you can tune them in a number of ways. The general objectives of tuning forms are the following:

- Reduce the load and run times for the form.
- Optimize database access from the form.
- Reduce memory requirements on the client.
- Minimize network traffic.

Whether you are deploying on the Web or in a client/server environment, many of the same tuning principles apply. The techniques that follow present Forms-specific tuning tips that will help both on a client/server and on the Web. There is an additional factor on the Web because of the multi-tier architecture and techniques for that environment are discussed in a section below on "Tuning Web-Deployed Forms." First, there are some tuning concepts worthy of an initial mention that lie outside the form.

TUNING FROM OUTSIDE THE FORM

Often, factors other than the programs that they are working on are out of bounds to the developer. There is usually someone other than the developer who handles hardware, network, and database installation and maintenance. However, all these factors affect how quickly a form runs and accesses the database, so the developer needs to be sure that these areas are being handled as efficiently as possible by whoever needs to configure them. The main areas outside the form that affect its performance are code tuning, client tuning, network tuning, database tuning, and application tuning.

CODE TUNING

Although it is beyond the scope of this paper to discuss SQL and PL/SQL tuning, these are important enough to mention. The code that you embed in the form has to be as efficient as possible so, to properly tune your entire system, you need to tune the SQL and PL/SQL statements that the form uses. Start this type of tuning with the SQL code because it will have the most effect. Inefficient PL/SQL code is usually only a minor contributor to system slowdowns. Once again, consult other references (such as books on the languages) to determine how to make your SQL and PL/SQL code the most efficient.

CLIENT TUNING

The client machine must have enough resources installed to execute the runtime program whether it uses the browser or the Forms client/server runtime. Although processor speed is important and you want to run on a machine with the fastest processor possible, the main other resource is memory. If a form runs out of memory, the operating system will use virtual memory. This means that there will be disk access to save and swap the contents of the memory space. Disk access takes much more time than memory access and can be avoided with a sufficient amount of memory. Pay attention to the minimum runtime requirements listed in the *Getting Started* manual and, if possible, double them. At this writing, the suggestion is 8MB of RAM for each client/server runtime session (in addition to the operating system requirements). With web deployment, the requirements are similar. In both environments, the more RAM available, the less the runtime engines will need to swap memory to disk, and the faster the application will run.

The minimum requirements for a development machine are much greater because developers often use many tools at the same time. The suggestion in the *Getting Started* manual is 16MB for each builder (for example, Form Builder) that you run. It is important to test the forms on machines that emulate the typical (or even minimum) user's setup. Testing forms using a development machine will not give you a true picture of what users will see if their installed memory and processor is different.

NETWORK TUNING

In the multi-tier web environment, there are network connections between client and middle-tier servers and between middle-tier servers and the database server. All these connections affect the speed at which a form runs. It is therefore important for the network connections to be as fast as possible. This involves both network hardware—interface cards, network lines, hubs, routers, and concentrators—and network software configurations—for user account access validation, domain scope, and network protocol and architecture (LAN or WAN, modem or faster connection). Most of this is clearly beyond the developer's duty, but the data in the form, and even the form itself, is communicated across the network, so the developer must be satisfied that all factors external to the form are working at peak efficiency.

If you work in a client/server environment, you can install Forms Runtime on a network server. The promise of this strategy is that there is less maintenance when the Forms version changes; only one machine needs to be upgraded. However, there are files that must be locally installed in the client's Windows directory, and there are registry settings required on the client side as well, so this benefit is reduced. In addition, running client/server forms from a network drive can cause major drains on the network. When the Forms Runtime is "downloaded" to the client machine's memory during startup, the network is transmitting many tens of megabytes of executable files and. DLLs. This process is much faster if the runtime files are located on the client machine.

With the multi-tier web deployment strategy, this effect is lessened significantly because the runtime files are located on the application server tier. The client only needs the browser and whatever Java support files are required to display the form running on the server. Tuning focus, in this case, shifts from the client and network file server to the application server and database server.

DATABASE TUNING

Other than the application logic that you write (discussed next), Forms' native access to the database requires no special database tuning, but it is important to be sure that data access in general is making the best use of the shared pool area so disk access is minimized. The Oracle8i cost-based optimizer usually chooses the most efficient access path for a SQL statement, but it requires a recent set of statistics that must be collected as a maintenance procedure (using the ANALYZE command) on a regular basis.

There are many other factors, such as INIT.ORA parameters and rollback segment sizes, that affect how efficient the database is across all of its applications. These subjects are really in the realm of the DBA. However, as before, this affects the efficiency of the form and is a factor that the developer must consider.

APPLICATION TUNING

Along with client, network, and database tuning, the design of the data structures for your applications critically affect their efficiency. If you denormalize columns (to store summary information, for example) and the denormalized column data is loaded at runtime, the system could slow down and the form will seem to be running slowly. If you are missing indexes that would speed a query by many times, the form queries will be affected. However, too many indexes can slow down a DML statement, so you have to be judicious in index use. These are only a few of the hundreds of factors that affect how efficiently your application accesses data.

Application tuning is usually performed only when there is an inefficient component. There are many tools you can use to determine how quickly a SQL statement is running and what path it takes to find the data. If a form appears to be running slowly, you can ALTER SESSION SET SQL_TRACE TRUE, as you would in SQL*Plus, by setting the Statistics preference (in the Reference tab of **Tools→Preferences**) to log the session to a trace file on the server. There is also a runtime parameter, STATISTICS, that accomplishes the same thing. The trace file will give you clues as to which statement is taking a long time and what the access plan is for the data.

The topic of designing efficient databases is the subject matter for many more white papers. The main tip here is not what kinds of application tuning to perform, it is that application tuning is key to how efficient the user perceives your forms to be.

> *TIP*
>
> If you want to improve the performance of a frequently used database package that you are calling from your form, use the DBMS_SHARED_POOL database package's KEEP procedure to load the package into the shared pool memory area of the database. The package will run faster if it is loaded from memory than if it has to be loaded from disk.

TUNING FROM INSIDE THE FORM

Once you have tuned the infrastructure as just mentioned, you can examine your form and see if there is anything that would make it more efficient. Many of these factors are merely tools in the developer's toolbox that should be part of normal form design and coding. The form-oriented tuning tips can be grouped into categories of General Tips, Form-Level Techniques, and Block Property Tips, and Forms PL/SQL Coding Tips.

GENERAL TIPS

The following tips provide general guidelines that you can follow for tuning your forms.

- **Partitioning the code** It is important for performance to store database access code in the database. This strategy will save network round-trips and allow you to place a one-line call to the database in your form to represent a block of code.
- **Modularization** The smaller your forms are, the faster they will load. There are always logical groups of objects in a form that you could consider making into separate forms that you call or open. However, you have to weigh the benefits of smaller Forms modules (shorter load time and less client memory) against the load time required for the forms that the user will call. For example, if you split FormA into two smaller forms, FormB and FormC, when FormB calls FormC, there is a short lag in time. If this call must be made many times in the course of a session, the accumulated wait time could frustrate the user. The guideline is that if there are parts of a large form that are used only occasionally, they should be considered as candidates for separate forms.
- **Use views to avoid database lookups** Consider a form built on the EMPLOYEE table that has a manager ID value. When you query this record into a form, you want to show the manager's name instead of the ID. A standard technique for accomplishing this is to write a POST-QUERY trigger that opens a cursor using the ID, fetches a record, closes the cursor, and loads the value into the non–base table item. This technique is inefficient because you have to open and close a cursor for each row retrieved in the query.

A better method is to create a view that contains the manager's name and use the view as a source for the block. You need to worry about DML statements, but you can use updateable views or INSTEAD OF triggers on the views to effect DML on the view. The savings is significant because the database will return all required values without any need for cursors and POST-QUERY processing.

- **Use a PL/SQL table to avoid database lookups** Another technique you can use for querying non–base table values for a lookup, is to store the names and ID numbers in a PL/SQL variable (or Forms record group) and use this as the source for the lookups. You can load the record group or table of records when the form starts, and use this source to look up the values in the POST-QUERY trigger. This method allows you to use a table as the basis for the block, but eliminates the cursor processing for each row. If you use a record group, you can load it from a query without having to write cursor loops. You can also find records using the GET_GROUP_RECORD_NUMBER built-in. If you use a PL/SQL table of records, the loading of the values requires a bit more programming. This technique is only viable for a small number of records (several thousand or less) because the variable arrays take up memory space.

  If the potential number of values you need to look up is small, you can load the values on an as-needed basis. For example, if you just need to look up the manager's name, instead of loading all manager IDs and names into a variable, you can query them when required in the POST-QUERY trigger. After querying an ID and name, you load it into the record group or table of records. When the next row comes in, the POST-QUERY trigger looks in the variable to see if there is a match. If not, it opens, fetches from, and closes a cursor for that row and stores the retrieved value in the variable. This technique is indicated only when the number of rows required in a certain query is small but the number of records in the entire table is large enough that you don't want to load it into a variable.

FORM-LEVEL TECHNIQUES

The following techniques fall into the category of form design, features, and properties.

- **Fewer objects** Using fewer objects in a form logically leads to modularization. There are things you can do, however, within the form to reduce the number of objects. If you are using stacked canvases to hide items, try setting the display property of the items instead. You can eliminate the canvas if you can use the item property. If you are using items on the null canvas to store variable values, use package specification variables instead because items, even though they are not rendered, take more memory because they have property values. The same principle applies to form parameters—avoid parameters unless you are using them to receive values passed into the form. Also avoid global variables because they use extra memory and are difficult to manage. Use package variables instead because there is less overhead.
- **Eliminate unnecessary boilerplate objects** Boilerplate text and graphics add to the size of the form. In particular, an embedded graphic on a canvas can add significantly to the .FMX file size. For example, a 20K graphic .TIF (Tagged Image File Format) file can increase the size of the .FMX file by 100K because the file is stored in an internal Forms format not as an optimized TIFF. The larger the FMX file is, the longer it takes to open initially.

  Alternatively, you can use the READ_IMAGE_FILE built-in to load image items from the file system. This requires an image file load, but the size of the file can be smaller if it is stored in a compressed graphics format (such as .TIF). This will eliminate the requirement for boilerplate graphics. Graphics drawn with the Form Builder drawing tools have slightly less impact on the canvas, but still take up enough room that the image file strategy is worth considering. An extension of this guideline is to use multimedia files sparingly, if at all. They take up much bandwidth and may not be totally required in some cases.
- ***Interaction Mode* property** The *Interaction Mode* property of the form module allows you to specify that the user can stop a query before it has completed. If you set this to the value "Non-Blocking", Forms will present a dialog box containing a Cancel button. If the user selects the Cancel button, the query will stop. However, there is overhead with this operation, and the query may take more time because of this. The best advice is to leave this property at its default value, "Blocking".
- **Forms Runtime Diagnostic** The Forms Runtime Diagnostic (FRD) is a method for capturing all events that occur in a form session and writing a file with the details of those events. There is overhead in writing this file, so you should only use this for the purposes of development and debugging.

BLOCK PROPERTY TIPS

The block is the main link to the database. There are a number of block properties that can assist in tuning.

- **DML Array Size** This property can reduce the network trips. If you set this to a number greater than the default of "1", the form sends an array of more than one record to the database when you commit a DML statement. This takes more memory on the client or middle-tier side, but can dramatically reduce the number of messages on the network. The best setting for this property is the number of records that require committing in one transaction. Since this will most probably be different for each transaction, you have to make an educated guess. Every row that is sent takes more memory, so there is a drawback in setting the number too high.
- **Update Changed Columns Only** The default value of this property is "No", which means that, regardless of what column values changed, all columns participate in the update. The UPDATE statement is made up of all columns with their current values. Setting this property to "Yes" might help if there are many items in the block but you usually update only a few of them (and always the same ones). If you only have a subset or a small number of columns in your form and you are not updating the same ones each time, this will not help because the SELECT statement will be different each time. If the statement is the same as one that was issued before, that statement might be found pre-parsed in the database's shared pool area. If so, the statement's execution will be more efficient.
- **Optimizer Hint** This property can specify that the optimizer use a certain access plan for a set of data. You can change this programmatically if you need different optimization paths for different situations. For example, you want the optimizer to use a certain index, but you have determined by using SQL_TRACE and examining the plan that it is not using the index. You can enter a hint in this property, and Forms will use this when it accesses the table. While hints for normal statements are enclosed with comment markers ("/* */"), you don't need to use comment markers in this property.
- **Query Array Size** The number of records fetched at one time can affect the perceived response time. The default value is "0", which indicates that the array size is the same as the number of records displayed. Setting this property higher can reduce the network trips, but will take more memory on the client or middle-tier side. You set this property higher mainly for multi-record blocks where users will most probably scroll after seeing the initial set of rows. Setting it higher for single-record blocks will make the form appear a bit slower, but this effect may be worthwhile if the user will probably scroll to another record after the first record. Since a low number increases the network traffic, you should try to strike a balance between setting this property value too high and setting it too low. Experimentation may be the only way to tell what the exact setting should be. When in doubt, use the default of "0".
- **Number of Records Buffered** The default for this property is "0", which means that the number of records in the block that are held in memory is equal to the number of records displayed plus three. If the user scrolls back and forth through the block (usually more common with multi-record blocks), you want to set this higher. The more records held in memory, the faster the records will reappear in the form when the user scrolls. Records that exceed this setting are buffered to disk. Therefore, the form always stores the records and does not reissue the query to the database. However, even disk access of records swapped to disk can take a noticeable amount of time. Therefore, set this property high enough that the user will be able to scroll without a lot of disk activity, but low enough that the available memory is not taxed by being filled with record data. The size of a record also plays into this consideration because you will use more memory for larger records.
- **Query All Records** You usually want to leave this property at its default of "No". This means that the block will initially query only the number of rows set in the *Query Array Size* property, which is faster than returning all records. The benefit for setting this property to "Yes" is that you will be able to calculate summary values of all the rows. You also need to set this to "Yes" if you have calculated items and the *Precompute Summaries* property is set to "No". The *Precompute Summaries* property issues a SELECT statement requesting the values of the calculated items before the query is sent. One of the two properties has to be set to "Yes" if the block contains summary items. If a query could return a large number of rows, you should set this property to "No" so that the user does not have to wait for all rows to be retrieved before working with the form. If there are few rows, then "Yes" is a reasonable value for this property.

FORMS PL/SQL CODING TIPS

There are several techniques that apply to the PL/SQL code that you write.

- **Use Libraries** This is common practice with most developers, but using .PLL libraries provides you with code sharing and memory sharing. If many forms access the same library, the memory that the library takes at runtime will be shared. This means that loading the code in the library will be faster because it is already in memory.
- **Use Find built-ins to get the ID of an object**. For example, you can use the following code to set an item property:

```
SET_ITEM_PROPERTY('EMP.EMPNO', VISIBLE, PROPERTY_TRUE);
```

When this line is executed, the form must look up the ID of the item EMP.EMPNO and apply the property setting to that item. You can use FIND_ITEM to find the item ID and issue that ID to the SET_ITEM_PROPERTY built-in. This is slightly more efficient and may be noticeable if you are setting a number of properties for an object. The revised code is a bit longer as shown here:

```
DECLARE
   v_item_id      ITEM;
   v_item_name    VARCHAR2(61) := 'EMP.EMPNO';
BEGIN
   v_item := FIND_ITEM(v_item_name);
   IF ID_NULL(v_item)
   THEN
      message('Error finding ' || v_item_name);
      raise form_trigger_failure;
   ELSE
      SET_ITEM_PROPERTY(v_item, VISIBLE, PROPERTY_TRUE);
   END IF;
END;
```

This code also incorporates error checking to be sure that SET_ITEM_PROPERTY will not be called if the item does not exist. This sample could be easily genericized by placing it in a packaged procedure that accepts a parameter of the item name. Other objects such as windows and canvases have similar FIND_ functions. The help system topic "Referencing Form Builder objects by internal ID" describes this in detail. (You can find this topic in the "related topics" list on the FIND_ITEM built-in description.)

- **Use a PL/SQL variable instead of a bind variable.** Referencing a bind variable that represents a forms item (such as :DEPT.DEPTNO) requires forms to find the item in its internal list. Therefore, each time your code uses that same bind variable, the lookup has to occur again. If you have code that uses the same bind variable more than once in the PL/SQL block, create a variable instead. For example, the following code stores the value of :DEPT.DEPTNO in a variable and references that variable instead of repeating a lookup to find the value.

```
DECLARE
   v_deptno    NUMBER := :dept.deptno;
BEGIN
   message('The department # is ' || v_deptno);
   IF v_deptno < 0
   THEN
      message('Incorrect department number');
      ...
   END IF;
   --
   :control.department_num := v_deptno;
END;
```

TUNING WEB-DEPLOYED FORMS

The preceding discussion has focussed on general techniques that will improve your Forms application whether it is running in client/server or on the Web. There are a number of specific techniques that you can use to optimize the form specifically for web deployment. Since this is a popular style of deployment currently, some of the general tips are repeated with a focus on how they improve performance on the Web. In the following discussion, there is mention of the *Forms Server*, which has been renamed to *Forms Services*. Forms Services are part of the Oracle9i application server. The previous release of Forms Services was a separate product—Forms Server. Both terms refer to the Forms application server process despite the change in packaging.

> *NOTE*
>
> This author's paper "Top Tips for Web-Deployed Forms" on the IOUG-A Live 2001 conference proceedings CD repeats some of these tuning tips and provides additional information on web-deployed forms.
>
> OTN provides a white paper on tuning web-deployed forms that is referenced in the "Other Readings" section at the end of this paper.

SIMPLIFY YOUR FORMS

Forms that have a small number of GUI items work very well. In contrast, forms with many items on one canvas can be problematic. Performance is reasonable with a limited number of users (based on the power of the application server) and a dedicated server with a basic level of memory (3–10 megabytes (MB) per concurrent user in R.2 and 1–6MB in R.6). Oracle has tested memory usage on the application server of a small form to be 1–2MB per user. They can support between 100 and 500 users per CPU and have proven the viability of web-deployed forms with up to 5,900 concurrent users.

You can also try to avoid intensive operations that are more network-intensive than others, such as programmatically changing visual attributes; using current record attributes; and changing item properties for size, position, enable mode, visibility, labels, and prompts. Menu item actions (enabling and disabling) are also network intensive. Of course, these are all key features of Forms so you will not be able to eliminate their usage completely.

> *CAUTION*
>
> If you use the strategy of creating many forms instead of one large form, the initial startup of each form will be faster, but moving from one form to another may be slower because a new form needs to start up. If the functions on the other form are rarely used, dividing the form may be better. Most of the time, the form would load quickly. But when you call another form, there would be a longer delay than usual, because a new form is starting. This wait may be less objectionable to users than the initial startup wait, so this strategy may prove useful.

REDUCE NETWORK TRAFFIC

This goal was mentioned before. There a number of tips that affect the efficiency of your forms and that you should work into your initial design. Reducing network traffic is the name of the game for tuning web-deployed forms. Most of the tuning effort required is between the application server and client, as this is the most complex connection and the one that is most unfamiliar to Forms developers. It is useful to review some guidelines for reducing database server network traffic.

OPTIMIZE THE CLASS FILE LOADING

The initial load time of the Java applet can be a sore point with users because it seems longer than the form startup in client/server environments. Part of this time is spent in loading the Java class files on the client side. You can optimize this task using the following tips. The following list refers to Java Archive (JAR) files that are collections of the Java classes used to

render objects. You can create and open and manage these files using an archive utility such as WinZip. Most classes are named descriptively, so you can check whether a particular function is in a particular JAR file.

- **Load a smaller JAR file** JAR files are cached if you use JInitiator. This helps reduce the initial download time to a minimum. Use the ARCHIVE parameter in your starting HTML file to load one or more JAR files. For example: PARAM NAME="ARCHIVE" VALUE="f60web.jar,icons.jar". The F60WEB.JAR file contains all but the LOV classes and is the one to use if you want to load all common classes when starting up.
- **Tune the Cache** The default Java memory cache on the client is 20MB. Compare that number with memory as the application is running (using a system monitor such as the Windows NT Task Manager), and ensure that you have enough cache space available in memory. Increasing physical memory will help if you are running out of memory and swapping to disk. Freeing memory by closing other applications will also help.
- **Locate the JAR file centrally** If JAR files are housed on different application servers in a load-balancing arrangement, the same JAR files could be downloaded from different servers because the classes are cached relative to the server. Therefore, locate the JAR files centrally if you have a load-balancing configuration.
- **Use the Deferred-Load Feature** A deferred-load feature allows you to embed references to other JAR files within a JAR file. The referenced files will not load immediately, but will wait until the class in that file is required. This deferred loading means that the form can display faster initially, although other classes that are required will be loaded as needed. Oracle supplies a number of JAR files with different contents, but you can create your own JAR files (using an archive utility such as WinZip) that contain references to other files.
- **Store the .GIF icon files in the JAR file**. This saves download time. .GIF files are used for icons on buttons in web-deployed forms instead of .ICO files. Some icons are actually supplied by the Runtime engine and do not require .GIF files at all. There is no known list of these icons, but you can experiment with the icons that are used by the default Forms toolbar (exit, save, etc.). The images will display even if you delete those icons from the icon directory.
- **Store beans in a JAR** If you are using Java code (JavaBeans or Pluggable Java Components), store them in the JAR file as well to speed up the download.

> *CAUTION*
>
> Regardless of the tuning benefits mentioned, creating your own JAR files is a bit difficult and is normally unnecessary. The JAR files supplied with Oracle Forms Developer will serve you well for most situations. In addition, Oracle discourages modification of the supplied JAR files. If you find that you need to modify the contents of a JAR file, it is best to create your own JAR file. That way, if Oracle upgrades a supplied JAR file, you will not have to redo your JAR file work.

REDUCE BANDWIDTH USAGE

The key factor that differentiates web-deployed forms from client/server forms is their use of the network. Anything you can do to reduce network traffic (that uses available bandwidth, or capacity) will speed up the way the forms load or run. The following points address this principle.

- **Reduce the number of boilerplate text objects**, and use the prompt property instead. Boilerplate text is another object that needs to be rendered.
- **Reduce the number of boilerplate graphics**. Lines and rectangles are optimized, but other types are not.
- **Change navigation** If your form contains many windows, allow the user to navigate to those other windows by clicking an OK or Done button instead of by pressing TAB to navigate through items. If the user does not need to change anything in those items, it is a waste of time to have to navigate to them. In addition, more triggers will fire as you navigate through items (such as POST-TEXT-ITEM and WHEN-NEW-ITEM-INSTANCE). If the user can skip to the next window with a button, the network traffic required by the item triggers will be eliminated.
- **Use a simple startup form** with just a few items on it. An example would be a logon screen with only a few items and graphics. This loads faster.

- **Hide objects not initially required**. Set the canvas property *Raise on Entry* to "Yes." Set other objects' *Visible* property to "No." Also set *Raise on Entry* to "Yes." Neither of these are the default values. When the cursor navigates to the item, Forms will automatically display the canvas. You can also issue a SHOW_VIEW call to set the *Visible* property to "Yes" programmatically. Tab canvases load all objects for the entire set of tabs at the same time. Set the *Visible* properties of all items to "No" to counteract this. Set them back to "Yes" when you navigate to a particular tab.

OTHER TUNING TIPS

Here are some other techniques you can use to tune the runtime environment:

- **Avoid repeating timers** Timers generate extra network traffic, as a packet is sent each time that the timer expires. If you destroy the timer and do not allow it to repeat, there is no adverse effect. Instead of using repeating timers, use JavaBeans (or Pluggable Java Component—PJC) with the same functionality. JavaBeans will fire on the client side and no extra network traffic will be generated. The forms demo files include some sample files that help you get started. These files are located in the ORACLE_HOME\FORMS60\java\oracle\forms\demos\source directory. There is a file called PJCREADME.TXT file in that same directory that contains notes on the sample PJCs. In addition, the OTN website contains a sample called "ScrollingTicker JavaBean" that demonstrates how to include a repeating timer application on the client using a JavaBean.
- **Avoid MOUSE-UP and MOUSE-DOWN triggers**, as these require extra processing on the server. Both triggers are handled by the same Java event. The Java event sends a message to the Forms Server regardless of the situation (MOUSE-UP or MOUSE-DOWN). The Forms Server needs to determine which one (MOUSE-DOWN or MOUSE-UP) is appropriate when this Java event occurs. Therefore, if you have a MOUSE-DOWN trigger but no MOUSE-UP trigger, the Java event will occur even in a MOUSE-UP situation. This is extra work for the Forms Server if you are not using that trigger.
- **Avoid high-level triggers** For cleanliness of coding, you may have written block- or form-level triggers, for example, a block-level trigger on a toolbar block that checks which button was clicked. The problem with high-level triggers such as this is that the trigger will fire for each object regardless of whether you are processing the object in the trigger. Each trigger firing will cause extra network traffic. It is more efficient to use item-level triggers on only the items that require the code and remove the higher-level triggers.
- **Simplify validation** Item validation generates application server messages and processing. Use Java plug-ins instead of standard Forms widgets to do validation. For example, you can replace a text item with a Java item that already contains validation code so that the validation will occur on the client and save network traffic. The extra code may require extra time to load the form, but this disadvantage may be less important than the savings in validation time.
- **Use fewer displayed items** on each canvas. Many items on a canvas means that there are many objects to initialize, and this will take more time to display than a canvas with fewer items.
- **Use NEW_FORM** instead of OPEN_FORM or CALL_FORM. Multiple forms open at the same time take more memory on the client. If another user has the same form open, it will share the memory on the server that is used by the program units. (No data portions or variable values of the form are shared.) However, if the user needs to return to an form that was previously open, you might consider using OPEN_FORM so that the reload does not require restarting the form, but just switching focus to it.

  One guideline is to use OPEN_FORM when memory is not a problem on the server and client and the network latency is high (i.e., the speed is slow). Use NEW_FORM when there is limited memory on the server and client and there is low network latency (a fast network).
- **Use subclasses** The "message diff-ing" feature of the Forms Services reduces the messages required to draw objects on the form by caching instructions for objects that share similar properties. For example, if you have an item that is 16 points high with an MS Sans Serif, 9 point font, the next item that the form needs to draw with those same characteristics will not require reloading the instructions from the server—they will be cached on the client because they were used once. Using default values for properties, subclassing (and SmartClasses) and visual attributes all result in standard settings that will allow the message diff-ing feature to work.
- **Locate the Forms Services (server) physically close to the database server**. The reason is that the internal tuning that Oracle has performed on the Forms Services to client connection solves problems of network slowdowns. The

database connection from Forms Services uses standard SQL*Net (Net8) mechanisms and this has no special tuning possibilities for Forms other than reducing the physical distance between machines.

BENCHMARKING YOUR TUNING

It is important to be able to document how your tuning efforts improved the performance. The general idea is that you want to capture the execution time of problem areas before and after tuning. Comparing these timings tells you how effective your tuning was. There are several ways to benchmark the form's performance.

PERFORMANCE COLLECTOR

The Performance Collector is a feature of the Forms Runtime Diagnostics (FRD) utility in Forms 6i that is useful in client/server as well as web-deployed environments. It automatically stores into an ASCII text file the timing of events such as when the Forms Services receives the request to start a session, when the Forms Services responds to the request, when a logon occurs (requested and received), when a query occurs (requested and received), and when a logoff occurs.

You start the Performance Collector using the RECORD command-line parameter RECORD (for example, RECORD=PERFORMANCE). For normal FRD output, you use RECORD=COLLECT. Here is an example of a command-line that starts up the Performance Collector:

```
ifrun60 module=testform.fmb userid=scott/tiger@orcl  record=performance log=testform.log
```

If you are running forms on the Web, this string (after ifrun60) is embedded in the startup HTML file in the ServerArgs section. The output is readable, but not formatted. There is a Perl script (f60parse.pl) that you can use to format the output. This file is located in the ORACLE_HOME\forms60\perl directory on whatever machine Forms runtime is installed.

The OTN white paper mentioned in the "Other Readings" section of this paper provides details about this utility.

OEM FACILITY—ORACLE TRACE

Oracle Trace, (available through Oracle Enterprise Manager (OEM) with the Diagnostics Pack) allows you to track various forms events. This feature is available starting with Oracle Trace release 2.1. The utility will track the timings of many Forms actions (such as session, form, query, LOV, trigger, SQL, logon, logoff, etc.) and the interaction with various objects (alert., editor, window, canvas, etc.). Some of these actions are the same as those tracked by Performance Collector, but the OEM solution is more complete.

Oracle Trace is available outside of OEM. When you install the Forms Services, there are executables installed that provide a manual way to accomplish what OEM does through its UI. This utility writes two log files that must be formatted after the form is run. The OTN white paper mentioned in the "Other Readings" section of this paper contains the steps you use to run Oracle Trace on your forms with OEM and without OEM.

ORA_PROF

The ORA_PROF built-in package (documented in the Forms help system) allows you to create, destroy, and restart timers in your forms. These functions are all possible with the standard timer built-ins, but ORA_PROF also contains a function called ELAPSED_TIME that allows you to query the current time that has elapsed since the timer started. This function is not available through the standard Forms timer built-ins. You can use this package for displaying the elapsed time between operations in your form. Writing this information to a text file using the TEXT_IO built-in package can provide a reference for how beneficial a tuning effort has been.

> *NOTE*
>
> The Performance Event Collection Services (PECS) form that was included with earlier Oracle Developer releases has been discontinued. Earlier versions of Forms, such as 6.0 use a command-line parameter PECS (PECS=ON) to activate performance tracking.

OTHER READINGS

The following are white papers and other readings that you can refer to for more information on the concepts discussed in this paper. The papers that are marked as "OTN" are currently on the Oracle Technology Network website (http://technet.oracle.com). The OOW papers are on the Oracle OpenWorld website (http://www.oracle.com/openworld, follow the link to search for white papers from OOW 2000).

- *How To Tune Your Oracle9iAS Forms Services Applications*, This OTN white paper contains detailed information about the new features of Developer R.6 and how you can take advantage of them. Also available as OOW white paper 296 by Regis Louis.
- *Configure and Deploy Oracle Forms Developer Server 6i on the Web*, OOW white paper 473 by Becca Martin. This describes the Forms Server (Forms Services) in detail and provides some tips for tuning the application as well as the server.
- *Guidelines for Building Applications - Performance Suggestions*, Forms online documentation (**Help→ Manuals**). This file is located at ORACLE_HOME\tools\doc60\us\guide60\perfmnce.htm.
- *Performance Collector for Oracle Forms 6i*, an OTN white paper that describes details about the Performance Collector.
- *Oracle Trace Integration with Oracle Forms 6i*, an OTN white paper that covers the Oracle Trace facility in Oracle Enterprise Manager (OEM)
- *Oracle9iAS Forms Services Diagnostics Techniques*, this OTN white paper (also called "Forms Diagnostic Techniques") contains a section on "Performance Problems" with solutions and things to check.
- *Oracle Forms Server Troubleshooting*, an OTN white paper that describes troubleshooting in general. It also contains a section on the Forms Runtime Diagnostics that you can use to benchmark and troubleshoot performance issues.

ABOUT THE AUTHOR

Peter Koletzke is a Technical Director and Principal Instructor for the Enterprise e.Commerce practice at Quovera (formerly Millennia Vision), in San Jose, California. Peter is Executive Vice President and Director of Web Initiatives for the IOUG-A and columnist for the *ODTUG Technical Journal*. He is a frequent speaker at various Oracle users group conferences where he has won awards such as Pinnacle Publishing's Technical Achievement, ODTUG Editor's Choice, and the ECO/SEOUC Oracle Designer Award. He is the coauthor, with Dr. Paul Dorsey of the Oracle Press (Osborne McGraw-Hill) books: *Oracle JDeveloper 3 Handbook*, *Oracle Developer Advanced Forms and Reports* (which supplied material for this paper), *Oracle Designer Handbook, 2nd Edition*, and *Oracle Designer/2000 Handbook*. http://ourworld.compuserve.com/homepages/Peter_Koletzke

Quovera provides strategy, systems integration, and outsourced application management to Fortune 500, high-growth middle market and emerging market companies. The firm specializes in delivering intelligent solutions for complex enterprises, which improve productivity within the customer's business and optimize the customer's value chain, through integration of its customers and suppliers. The company also outsources the management of "best of breed" business applications on a recurring revenue basis. Quovera refers to its business model as "Intelligent – Application Integration and Management. http://www.quovera.com