

Tuning Developer/2000 Application Performance on the Web

November 1997

Tuning Developer/2000 Application Performance on the Web

November 1997

Author: Chris Hughes

Copyright Oracle Corporation 1997

All rights reserved. Printed in the U.S.A.

This document is provided for informational purposes only and the information herein is subject to change without notice. Please report any errors herein to Oracle Corporation. Oracle Corporation does not provide any warranties covering and specifically disclaims any liability in connection with this document

Oracle is a registered trademark, and Developer/2000 is a trademark of Oracle Corporation. Other trademarks acknowledged.

Tuning Developer/2000 Application Performance on the Web

This document is designed to explain how to optimize the performance of a Developer/2000 Forms application when deployed to the Web, and explain some of the technology upon which our 100% Pure Java solution is based. It is recommended that this document be used in conjunction with the documentation & release notes provided with the software.

- Optimizing Applet download
- How to determine which class files are required for an application
- How to completely avoid Applet download
- What is a JIT?
- Tips on application design for the Web

This document will assume that the reader is familiar with the Developer/2000 Forms product.

Optimizing Applet download

Background

The Developer/2000 Forms Server contains components for both the client & application server tiers of a three-tier architecture. The client piece of this architecture being the generic Java Applet, which performs the task of displaying the application to the user, and passing user interaction back to the application server.

A Java Applet can be thought of as a platform-independent program, which is made up of one or more Java class files. In the case of the generic Java Applet supplied as the client portion of the Developer/2000 Forms Server, the complete Java Applet is formed from over 100 distinct Java class files. The reasoning behind this seemingly large number of Java class files, being that our Java Applet separates distinct pieces of functionality into separate Java class files in a truly object-oriented manner. For example, if an application which contained no buttons (an extreme case) were deployed on the Web, then the Java Applet would have no use for the "ButtonItem" Java class file (and downloading this functionality to the client machine would be an unnecessary overhead).

Typically, the Java Applet is downloaded on-demand from the application server via a Web Server. The cue for the Applet to be downloaded to the client being the inclusion of an <applet> tag within an HTML file, such as:

```

<html>
<applet codebase="/ web_code/"
        code="oracle.forms.uiClient.v1_4.engine.Main"
        width=200 height=100>
<param name="serverArgs" value="fndscsgn apps/apps@gl80">
</applet>
</html>

```

Problem

In the example HTML file above (and many current customer configurations), only one Java class file will be initially downloaded to the client machine (the "Main" class file). The "Main" Java class file is simply the "entry-point" for the Java Applet, and to run any Forms application will require that more of the Java class files which make up the complete Java Applet are downloaded to the client machine. However, since the example HTML file does not identify what additional Java class files are required, the Java Applet will make a separate request to the Web Server for each & every additional Java class file required.

So, in the case where an application uses 70 of the complete 100 Java class files supplied, specifying the <applet> tag as above would result in 70 separate download requests being sent to the Web Server, which will significantly impact the perceived performance of the application.

Solution

One of the features introduced in the 1.1 release of the Java Development Kit was the capability to combine multiple Java class files into a single file with a portable format. This capability is provided through the JAR (Java ARchive) utility.

Making use of this utility, some or all of the 100+ distinct Java class files that make up the Developer/2000 Forms Applet can be combined into a single "jar" file. The great advantage of the "jar" file being that it provides a mechanism to download multiple Java class files to the client machine while only making one request to the Web Server.

Making use of a "jar" file simply requires a minor modification to the <applet> tag within the HTML file, such as:

```

<html>
<applet codebase="/ web_code/"
        archive="f45all.jar"
        code="oracle.forms.uiClient.v1_4.engine.Main"
        width=200 height=100>
<param name="serverArgs" value="fndscsgn apps/apps@gl80">
</applet>
</html>

```

The "jar" file referenced above, f45all.jar (shipped with the product), contains all 100+ distinct Java class files that make up

the complete Java Applet. This “jar” file may well contain Java class files that are not used by your application, but ensures that no additional requests are made to the Web Server. Using the jar utility, the list of files combined in the “jar” file can be customized to match the requirements of your application.

How to determine which class files are required for an application

Background

As discussed above, making use of a “jar” file can very significantly improve the perceived performance of an application, by minimizing the number of requests made to the Web Server. While the use of a “jar” file will always improve performance, the “jar” file used above, f45all.jar, may contain Java class files which are never used within your application. To gain the most benefit from the JAR capability, it is wise to customize the “jar” file to contain only those Java class files that are required by your application.

Problem

Without knowing internal implementation details of the generic Java Applet, how can I determine which distinct Java class files to place in my “jar” file?

Solution

As mentioned above, if you don’t make use of a “jar” file, each & every Java class file that is required by your application is downloaded from the Web Server separately. We can use this to our advantage to determine exactly which Java class files to place within our customized “jar” file.

By default, the Web Server will write out each request made, to its log file. So, to determine which Java class files an application makes use of:

- Temporarily remove the “archive” parameter in the <applet> tag within your HTML file.
- Once the application is running, navigate around the application, to ensure that all screens and functionality have been visited.
- Upon completion of this exercise, the Web Server log file will contain a complete listing of the Java class files that were requested for download.

To build a “jar” file which contains only those class files required for your application, follow the steps outlined below.

- Copy the complete Java class file directory structure (the “oracle” directory and all its sub-directories), which following a default installation on a Windows NT 4.0

machine will reside under %ORACLE_HOME% in the \forms45\java directory, to a new location.

- In the copied structure, remove any Java class files that were not listed in the Web Server log file.
- Archive the Java class files in this copied structure by issuing the following command from the new directory, which contains the “oracle” directory
 - `jar -cvf <"jar" file name> oracle`

which will recursively go down the “oracle” directory and place all the Java class files found into a new “jar” file with the name specified above. This new “jar” file can then be referenced in the <applet> tag of your HTML file by updating the “archive” parameter.

How to completely avoid Applet download

Background

In order for a Java Applet to be run, the Java class files that make up the Applet must be available to the Java Runtime Environment on the client machine. The Java class files required for an Applet are typically downloaded (either individually or within a “jar” file) on-demand from a Web Server.

While this technique simplifies the client-side installation, it does require that the Java class files be downloaded from the Web Server – which will impact the perceived startup performance of the application.

Problem

Users that have poor network connections to the Web Server observe that it can take up to two minutes before they are presented with the entry screen to an application. Making use of a “jar” file will help to reduce this time, but it may still take a long time to startup the application.

Solution

Installing the Java classes that make up the Applet on the client machine will significantly improve the perceived performance of the application, since the time taken to contact the Web Server and download the Java class files has been avoided.

The use of local class files is the optimum solution, today, for access using a slow speed line, such as a 28.8 modem. In the future, as browsers and the AppletViewer expose Jar and class file caching mechanisms, the need for local class files will be eliminated.

In order to install the Java class files on the client machine simply copy the directory structure (and Java class files) located, following a default installation on an Windows NT 4.0 application server, in

- `%ORACLE_HOME%\forms45\java`

to a new directory on the client machine. Once the class files have been copied, simply set the environment variable

- `CLASSPATH`

to reference this directory.

For example, if my client machine is a PC, the following settings could be used

- Create new directory as `c:\applet`
- Copy class file directory structure into `c:\applet`
- `set classpath=c:\applet`

The user can then access the same HTML file as previously used, and although the HTML file contains the same `<applet>` tag, the AppletViewer will use the local files found via the `CLASSPATH` environment setting rather than downloading from the Web Server.

What is a JIT?

Since its inception, Java has been designed to be a cross-platform technology, capable of running on many differing client machines. As a result of this design decision, the Java class files that make up a Java Applet or Application must also be capable of running on differing client machines without needing to be recompiled.

As a result, when a Java Applet is being run on a client machine, the Java Runtime Environment on the client machine has to “interpret” the Java class files in use. The process of interpreting the portable code in a Java class file into a machine-usable format will take a small amount of time, which must be performed for each line of code within the Java class file.

The concept of JIT (Just-In-Time compilation) technology, which is commonly embedded into commercial browsers, takes the machine-usable code produced from the Java Runtime Environment and stores it in memory. The next time that the same piece of code is executed, the machine-usable version of the code stored in memory is used, thereby avoiding the overhead of interpreting the Java code again.

If your client-side environment provides JIT compilation technology, it is recommended that you enable this. For example, within Microsoft’s Internet Explorer 4.0, select the “Internet Options...” menu item from the “View” menu, and

within the “Advanced” tab check the “Java JIT compiler enabled” checkbox.

Tips on application design for the Web

Images

Prior to release 1.6 the display of images, either image boilerplate or image items, is achieved by sending down the information to the client Java Applet pixel-by-pixel. For large images, this can result in a very large amount of information being sent across the network, so it is wise to review use of images within the application. In release 1.6, this area has been re-implemented to avoid the pixel-by-pixel approach, and download the image through an URL.

Canvas Properties

The traditional client/server version of Developer/2000 Forms gained its portability by making use of the internal Oracle Toolkit API to handle the displaying of windows, buttons etc. When deploying a Developer/2000 Forms application on the Web, instead of using the Toolkit API to display user interface on the local machine, all user interface requests are written into a message buffer, which is sent to the client Java Applet.

The **current** implementation of the application server does not remove information from the message buffer that will be sent to the client. As a result of this, whenever the application makes a request to show, for example, a canvas - this message is written into the buffer even if the same object is hidden again before the screen is synchronized.

While the occurrence of trigger code such as:

```
Show_View(' My_Canvas ');  
Hide_View(' My_Canvas ');  
Synchronize;
```

is not common, there is a commonly overlooked property of a canvas that plays a factor in this area. The “Displayed” property of a canvas governs whether display information for the canvas is placed in the message buffer when the parent window is initially shown – irrespective of whether the canvas

Review the property settings of your canvases, paying particular attention to the setting of the “Displayed” property. The following combination of settings for your canvases should ensure consistent behavior without placing unnecessary information in the message buffer:

- Displayed = False
- Raise on Entry = True

Timers

Some applications make wide use of timers to implement, for example, a clock within the Forms application that updates every second. While this code will continue to function when run on the web, it does mean that you are forcing a round-trip between the Java Applet and the application server to occur once a second. On a slow network, this may well flood the network with information that really isn't critical for the application. Consider rescheduling the timer to fire less frequently, if appropriate, on the web:

```
If ( Get_Application_Property( User_Interface) = 'WEB' ) then
    -- Only update the clock once-a-minute on the Web
    Clock_Timer := Create_Timer('CLOCK', 60000, REPEAT);
Else
    -- Update the clock once-a-second for client/server
    Clock_Timer := Create_Timer('CLOCK', 1000, REPEAT);
End if;
```

or whether the timer is truly necessary for the application.