IBM WebFacing Tool Performance Update using WebSphere Application Server V5.0

Dean Henkel
IBM eServer Solutions Enablement
Rochester, MN

Updated September 2003

Table of Contents

Introduction 1

Expectations 2

Getting Started 2

Hardware Prerequisites for HTTP Server and WebSphere Application Server 3

Basic WebSphere Tuning 4

CUM and Group PTFs 4

WebSphere Application Server Initial Heap Size Setting 4

Memory Pool Setting 5

IBM WebFacing Tool Optimizations 7

WebSphere Development Studio Client Version V4 and V5 7

Delivering Optimal Performing IBM WebFaced Applications 8

JSP Batch Compiler 8

JSP Pre-Touch 10

What is JSP Pre-Touch? 10

When to use JSP Pre-Touch 10

When not to use JSP Pre-Touch 10

JSP Compilation 10

prepareJSPs 11

prepareJSPAttribute 11

prepareJSPThreadCount 11

Configuring the Pre-Touch Attributes 11

ByteCode Cache File 13

When to Use ByteCode Caching 13

Using the User Classloader Cache 14

os400.define.class.cache 14

Additional User Class Loader Cache Settings 14

os400.define.class.cache.hours... 14

os400.define.class.cache.maxpgms 15

When not to use ByteCode Caching 16

Just-In-Time compilation... 16

Additional Performance Enhancements 18

Version 5.0 of the WebFacing Tool.. 18

Compression... 18

Display File Record I/O Processing 18

Tuning the Record Definition Cache 19

Cache Management — Definition Cache Content Viewer 21

Cache Management — Record Definition Loader... 23

WSADMIN and WebSphere Application Server Scripts.................................... 24

WSADMIN 25

crtwasinst... 26

startServer, stopServer.. 26

Browser Caching 26

IBM HTTP Server Powered By Apache to Serve Static Content......................... 26

Add a Directory to the Web... 27

Change Permissions on the deployed application... 28

Set fileServingEnabled for the deployed application.. 28

Regen the WebSphere Plugin 29

Application Deployment Options 31

Deploying a new application... 31

Precompile all JSPs on development server.. 31
Copy the Compiled JSPs 31
Use the EARExpander tool repackage .EAR file 34
Network Connection Speeds and Compression.. 35
PC Client... 38
Timings 39
Conclusion.. 46
Special Thanks.. 47
Reference Web Sites... 47
Trademarks 48

Introduction

There has been a lot of excitement in the IBM® eServer™ iSeries™ community regarding IBM's new WebFacing Tool and its promise to quickly move 5250 green-screen applications to the Internet. It is not necessary to be a Java® programmer to get an application to the Internet, but one should have a good understanding about how web applications perform and how to make them perform well.

When looking at performance, there are several aspects to consider — how long does the application take to start up, how well does the application server perform at the server level, and perhaps the most important, how well does the application perform to the end user.

There are and will be differences in performance between a traditional green-screen application and a browser-based application. This is inherent to all browser-based applications and is a result of the introduction of additional networking components consisting of the client pc, Intranet/Internet connection speeds, network hops, browser interface, etc.

What does this mean for Web-enabled applications? Solution developers and customers need to consider several factors when creating and deploying a Web-enabled application. Some of these factors are:

? Realistic performance expectations — Green-screen applications versus browser-based applications.

? Adequate server hardware — It is important to remember that new workloads use system resources.

? Reliable fast Internet/Intranet connection — A slow pipeline can be a bottleneck.

? Adequate client hardware — This is to accommodate for the significant amount of work done by browsers.

This paper was written to assist solution developers and customers with performance issues of a Web-enabled version of their application, and to provide them with some tips on how to get the best performance possible.

This is the second edition of the original white paper. The first version focused on the improvements using WebSphere® Development Tools Version 4 to both WebSphere Application Server V3.5 and 4.0. This edition focuses specifically on deployment of an application deployed to WebSphere Application Server - Express V5.0 using the WebSphere Development Studio Client V4 and V5.

New enhancements to this paper include a more detailed approach on when and how to use the JSP PreTouch tool, packaging and deploying an IBM WebFaced Application with precompiled JavaServer Pages™, and using the IBM HTTP Server powered by Apache to serve static content. In addition to runtime performance, this edition includes topics on using the WebSphere scripts to decrease installation and server startup times. Issues that are not directly tied to runtime performance but impact performance perceptions and usability are also discussed.

Developers who want a comprehensive look at the IBM WebFacing Tool that includes an end-to-end look at the tool should consider reading the IBM Redbook, IBM WebFacing Tool: Converting 5250 Applications to Browser-based GUIs (SG24-6801-00).

Expectations

Traditional green-screen applications are very fast because the data transmitted is very small. In a worse case scenario for a green screen application in which every field were changed, a 2K Buffer transmission would occur.

Browser applications transmit more data. Examine any Internet web site, such as ibm.com, and it is very common to see over 50,000 bytes of data per second being transmitted from the server to a browser. Over a dial-up or other slow speed Internet connections, the transmission time on the wire for one page can be significant. Multiply that by multiple users and other applications using network bandwidth, and some organizations may find out that the 256K byte frame relay pipe is not sufficient.

Each type of interface has advantages and disadvantages. Typically, data entry speed will be superior on green-screen applications. The ability to type ahead several levels of screens for experienced users is something a browser cannot compete with. Browser-based application have the advantage of the more modern look-and-feel as well as allowing access to multiple device types, including wireless phones and handheld devices.

The IBM WebFacing Tool is an excellent way to quickly modernize a 5250-based application. Page transition times of one to three seconds are quite normal on adequate server hardware, a fast Internet connection (minimum of 1.5MB connection speed), and a client running a browser configured for static content caching. If the advantages of a browser are appealing, the IBM WebFacing Tool is a great place to start.

Getting Started

There are five different basic performance considerations:

? Server Hardware Prerequisites for HTTP Server and WebSphere
? Basic WebSphere Tuning
? IBM WebFacing Tool Optimizations
? Using IBM HTTP Server Powered By Apache to Server Static Content
? LAN Connection Speed
? Client Hardware

Each category is discussed in the following sections and should help in getting the most out of a Web-enabled application.

Hardware Prerequisites for HTTP Server and WebSphere Application Server

WebSphere Application Server has minimum server-side hardware requirements. Make sure that the system under test meets those requirements. For complete information on the various versions of WebSphere Application Server, see the WebSphere Application Server for iSeries Web site at:
ibm.com/eserver/iseries/websphere

The new WebSphere Application Server - Express V5.0 for iSeries has lowered the server-side requirements to 300 CPW with a minimum of 512MB RAM. This is basically enough to get WebSphere Application Server started and running a small application with a small number of users. NOTE: These requirements are just a minimum guideline. A good rule of thumb is to have a minimum of 512MB RAM dedicated just for WebSphere Application Server. This brings the system minimum RAM to 768M which will be much more acceptable to users on a dedicated system. This is particularly important if the Apache Integrated GUI is being used to manage a WebSphere Application Server - Express V5.0 server instance. This GUI is an easy way for customers to manage the WebSphere Application Server - Express environment. However, the Integrated GUI runs in a separate application server requiring additional system resources, and in particular, increased memory resources.

This is not to say that a system with 768M of RAM will run web application and have it perform well. There are many factors that vary from customer to customer such as additional workloads, number of users, and application complexity that will determine a true iSeries system minimum hardware specification. Readers interested in accurately sizing an iSeries server can use the IBM Workload Estimator tool for assistance. The Workload Estimator Tool can be found at:
www-912.ibm.com/servlet/EstimatorServlet

Basic WebSphere Tuning

A common question from developers is: “How can I tune WebSphere Application Server to make my application run faster?” This is fairly easy and straightforward for a Web-enabled application:

? Make sure the latest OS400 CUM Package is loaded on the server

? Make sure the latest Group PTFs for WebSphere, Database, Java, HTTP server are loaded on the server.

? Verify the Initial Heap Size Setting

? Verify that the memory pool used by the WebSphere application server has enough RAM

For additional details, refer to the WebSphere Performance Considerations Guide located at:
ibm.com/eserver/iseries/software/websphere/wsappserver/docs/WS40PerfCon.pdf

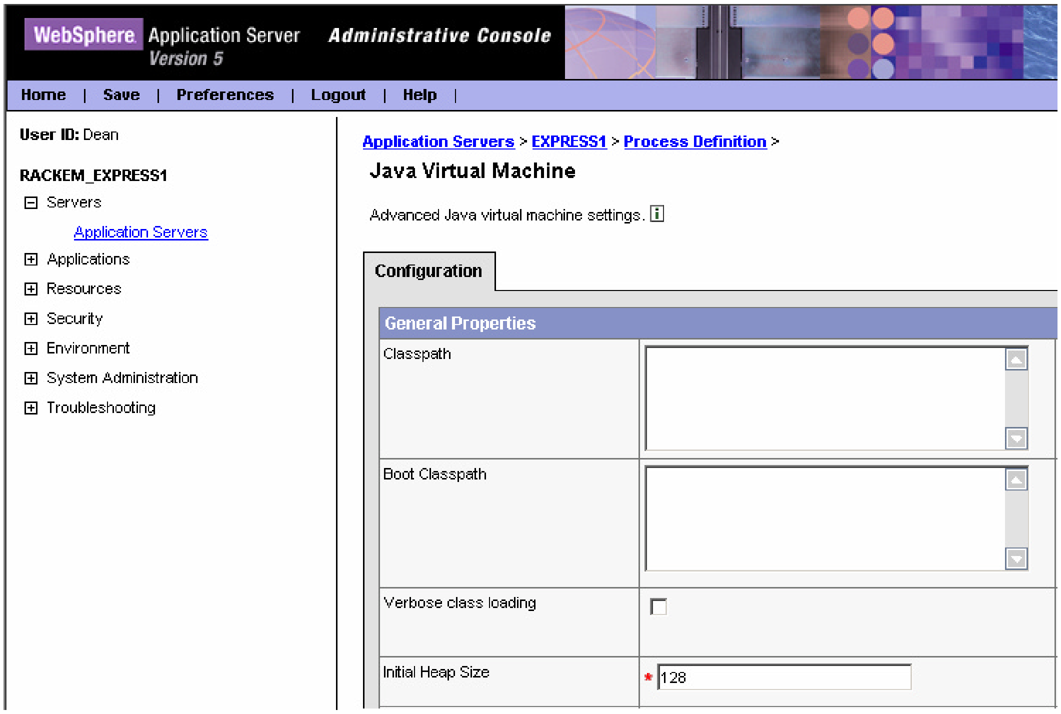
CUM and Group PTFs

Why load the latest CUM and Group PTFs? Simple, IBM is continuously making functional and performance improvements to these products. It is easy to take advantage of these improvements by installing the latest PTFs. To determine the latest PTFs for supported operating systems, check out **ibm.com**/eserver/iseries/support

WebSphere Application Server Initial Heap Size Setting

Setting the Initial Heap size is the single most important WebSphere Application Server setting that can help improve performance. Think of the initial heap size setting as the garbage collection threshold on an iSeries server. This means that when the free memory available in the heap is less than the initial heap size setting, garbage collection support will run to free memory. The setting varies based on the amount of memory in the memory pool, the number of processors, and the number of applications running within WebSphere. There is no real scientific way for setting the heap. Try different settings to see what works best. With the release of WebSphere Application Server V5.0, the default heap size setting was increased to 96MB. Previous versions of WebSphere Application Server defaulted the heap to 32MB which, in most cases, did not provide the best performance.

When using WebSphere Application Server V5.0, try the default setting first and see how the application performs. If performance is not acceptable, change the default heap setting to 128MB, “-Xms128m” — it may have a positive impact on user response times. For WebSphere Application Server V5 and WebSphere Application Server – Express V5.0, the heap setting is changed in the WebSphere Console under JVM Properties.



Application Servers > EXPRESS1 > Process Definition >

Java Virtual Machine

Advanced Java virtual machine settings.

General Properties

Classpath	
Boot Classpath	
Verbose class loading	□
Initial Heap Size	* 128

Figure 1 - Setting the Initial Heap Size using the WebSphere Application Server V5 Console

Setting the heap size too big may not yield any additional performance, but it could steal system resources and could actually cause the performance to worsen. Try different settings to optimize performance and system resources.

Memory Pool Setting

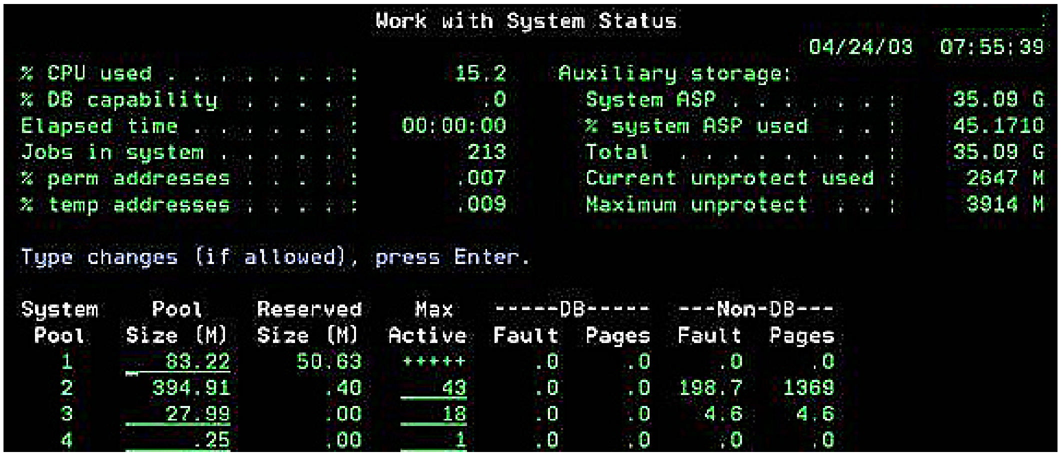
By default, WebSphere Application Server runs in the base memory pool, Base. Depending on what other applications are running within *BASE, it may not be necessary to setup a dedicated memory pool for WebSphere Application Server.

Users just starting to use WebSphere Application Server should leave the application server running in the base memory pool and see how it performs. If performance is acceptable, leave it alone. If performance is not acceptable, check to see how much memory is dedicated to *BASE — there may be that not enough memory has been allocated to *BASE to support WebSphere running in the pool. WebSphere Application Server needs a minimum of 512MB. Depending on the size and complexity of the installed application, it may require 1GB or more to the desired page response times.

Should it be necessary to allocate a dedicated pool for WebSphere Application Server, first check to insure that there is sufficient memory to reallocate from other memory pools. Improperly allocating and configuring memory pools can actually decrease performance. To learn about setting memory pools, visit the iSeries Information Center Web site at: http://publib.boulder.ibm.com/pubs/html/as400/infocenter.html

A very quick check can be made to determine if the pool WebSphere Application Server has sufficient memory. The WRKSYSSTS command displays system status. With the proper assistance level, users can view the page faulting rate occurring on the system. Using this command while running the application, observer the number of NonDB pages per second. As a rule of thumb, a pool averaging over 50 NonDB Page faults may mean more memory is need in the pool. Figure 2 is an example of heavy page faulting in *BASE. In the Machine Pool, the page faulting rate should be kept to an absolute minimum, generally less than five per second.

As shown in Figure 2, this system only has 512MB total system RAM. It is a minimum configured system with only 394M of memory in the pool where WebSphere Application Server is running. Running a small application under light load resulted in acceptable page response times. Running a larger number of users resulted in excessive NonDB page faulting. In this situation, increasing the memory size by adding additional RAM to the system would reduce faulting and improve page response times.



```
                            Work with System Status
                                                          04/24/03  07:55:39
 % CPU used . . . . . . . :        15.2   Auxiliary storage:
 % DB capability  . . . . :          .0     System ASP . . . . . . :    35.09 G
 Elapsed time . . . . . . :    00:00:00     % system ASP used  . . :    45.1710
 Jobs in system . . . . . :         213     Total  . . . . . . . . :    35.09 G
 % perm addresses . . . . :        .007     Current unprotect used :     2647 M
 % temp addresses . . . . :        .009     Maximum unprotect  . . :     3914 M

 Type changes (if allowed), press Enter.

 System     Pool    Reserved    Max    -----DB-----  ---Non-DB---
  Pool    Size (M)  Size (M)  Active  Fault  Pages  Fault  Pages
    1        83.22     50.63   +++++     .0     .0     .0     .0
    2       394.91       .40      43     .0     .0  198.7   1369
    3        27.99       .00      18     .0     .0    4.6    4.6
    4          .25       .00       1     .0     .0     .0     .0
```

Figure 2 - WRKSYSSTS Screen showing significant NonDB page faulting

IBM WebFacing Tool Optimizations

Consider IBM WebFacing Tool optimizations as one of the following:

? WebSphere Development Studio Client V4 or V5 with latest service pack
? Precompiling JavaServer Pages
? Creating and using a ByteCode Cache File
? Using IBM HTTP Server Powered By Apache to serve static Content

WebSphere Development Studio Client Version V4 and V5

The WebSphere Studio Development Client Version 4.0 contains many performance improvements over the earlier release of the IBM WebFacing Tool released in the Websphere Development Tools. Testing proved approximately a one-half second screen transition improvement compared to a similar application, using an earlier release of the tool. Dial-up connection performance improvements were even more significant.

Customers who are just starting with the IBM WebFacing Tool or are willing to migrate to the new tool should use the WebSphere Studio Development Client Version 5.0. From a runtime perspective, V5.0 provides equal or slightly better performance as compared to V4.0. There have been specific changes to optimize memory usage of the Webfacing runtime to improve scalability. However, these changes require that the caching support is configured optimally in order to achieve the best performance. If this is not done, V5.0 requires significantly more CPU than V4.0. Please refer to the section 'Memory Optimization for Record IO Processing' for how to configure the associated cache support.

In addition, compression support, similar to Apache mod-deflate, has been added to provide significant response times gains on slower internet connections. Refer to the section 'Network Connection Speeds and Compression' for more information and how to 'turn on/off' compression.
Regardless of what version is used, make sure the latest service pack is installed. Service packs can be found at:
ibm.com/software/ad/wds400/support/

More information about WebSphere Development Studio Client Version 4.0 & Version 5.0 can be found at: **ibm.com**/software/ad/wds400/

Delivering Optimal Performing IBM WebFaced Applications

JavaServer Pages (JSPs) are HTML pages containing embedded Java code capable of providing dynamic content. Unlike static HTML, the Java code within a JSP must be compiled into Java class files. The WebFacing Tool generates JSPs from the DDS source. Applications with large numbers of DDS screens will generate large numbers of JSP files. Each of these files needs to be compiled before they can be run.

When a screen change occurs, WebSphere Application Server makes sure that the compiled Java code is associated with the JSP. If it does, WebSphere Application Server executes the compiled Java code very quickly. If the compiled class does not exist, Websphere Application Server compiles the JSP file and any other JSP files that may be required to execute the request. This compilation is a very expensive operation at run time and can significantly delay the response time the first time a JSP is touched.

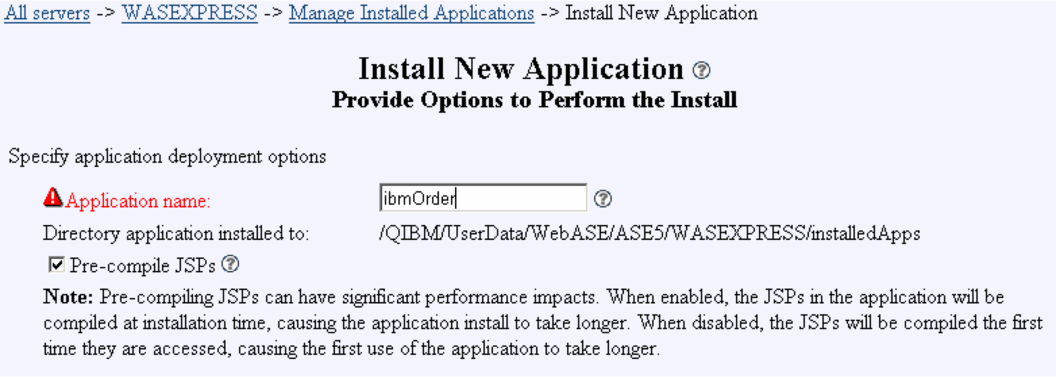
First impressions are very important. Ideally, an .EAR or .WAR file should be distributed to a customer with all JSPs precompiled. Creating an .EAR or .WAR file with precompiled JSPs is discussed in a later section of this paper. The following section describes how it is possible to compile and distribute the JSP files in an .EAR or .WAR file.

JSP Batch Compiler

WebSphere Application Server ships with two JSP batch compilers. The first batch compiler can be enabled using the Integrated GUI for WebSphere Application Server - Express or in the WebSphere Application Console and is an easy way to compile all of the JSP files.

The downside to the GUI version of the JSP Batch Compiler is that it is a synchronous operation at application installation time. This means that the installation process will not complete until all JSPs are compiled. This is a good option for small applications but can be cumbersome for large applications.

Figure 3 shows the JSP compiler option that can be checked during application installation.



All servers -> WASEXPRESS -> Manage Installed Applications -> Install New Application

Install New Application
Provide Options to Perform the Install

Specify application deployment options

Application name: ibmOrder

Directory application installed to: /QIBM/UserData/WebASE/ASE5/WASEXPRESS/installedApps

☑ Pre-compile JSPs

Note: Pre-compiling JSPs can have significant performance impacts. When enabled, the JSPs in the application will be compiled at installation time, causing the application install to take longer. When disabled, the JSPs will be compiled the first time they are accessed, causing the first use of the application to take longer.

Figure 3 — Precompile JSPs option checked during application installation

Information on how to compile JSPs using the batch compiler for WebSphere Application Server - Express V5 can be found in the WebSphere Information Center at:
http://publib.boulder.ibm.com/iseries/v5r2/ic2924/index.htm?info/rzamy/50/express.htm

For compiling JSP files when using WebSphere Application Server 4.0 information, see:
http://publib.boulder.ibm.com/was400/40/AE/english/docs/jsp11bcp.html

The second JSP batch compiler that ships with WebSphere Application Server is the script version. For WebSphere Application Server – Express, the JSP batch compiler is located in the \QIBM\WebASE\ASE5\bin directory on the Integrated File System (IFS) file structure. Using Qshell, this command can be invoked to compile all of the JSP files for an installed application.

When using the JspBatchCompiler script to compile an entire enterprise application, run the script from a temporary IFS directory. The script creates temporary files so running it from its home directory may cause problems if *RW permission has not been granted.

This is an example of running the JspBatchCompiler script from Qshell from a temporary directory that has *RW permission.

```
> /Qibm/ProdData/WebASE/ASE5/bin/JspBatchCompiler -enterpriseapp.name
ibmorder -webmodule.name ibmorder.war -instance MyInstance -server.name
MyInstance -verbose true
```

Running the same command in batch from an OS400 command script can be done as follows:

```
SBMJOB CMD(QSH CMD('cd /mycompile;/QIBM/ProdData/WebASE/ASE5/bin/JspBatchCompiler -
enterpriseapp.name ibmorder -server.name MyInstance -instance MyInstance')) JOB(BATCHCMPL)
```

Information on how to compile JSPs using the JSP batch compiler for WebSphere Application Server - Express V5 can be found in the WebSphere Information Center at:

http://publib.boulder.ibm.com/iseries/v5r2/ic2924/info/rzamy/50/program/jspbatch.htm

JSP Pre-Touch

WebSphere Application Server for iSeries has a unique feature known as JSP Pre-Touch. JSP Pre-Touch can be a confusing topic so let's try to simplify what it is, when to use it, and when to turn it off.

What is JSP Pre-Touch?

JSP Pre-Touch is a tool for compiling and loading JSPs during application startup. It is configured via a series of initialization parameters that are added to a web module's ibm-web-ext.xmi file and operates in a separate thread from the application server. Therefore, the application startup process is unaffected.

The reason that JSP Pre-Touch was created is to help reduce the amount of time it takes to invoke a JSP after an application is first started (called "first touch"), even when the JSP has already been compiled. The increased time during first touch is spent class loading the JSP class and optimizing the class (known as JIT, or Just-In-Time compilation). When the JSPs are class loaded, a step called byte code verification (which is quite costly) takes. The byte code verification time can nearly be eliminated by caching the results of verification using the byte code cache. Like other types of caching, however, the verification process must occur once in order to fill the cache.

When to use JSP Pre-Touch

The JSP Pre-Touch mechanism should be used when all JSPs need to be compiled asynchronously and/or the cache file needs to be primed.
When using the JSP Pre-Touch tool to prime the byte code cache, it is highly recommended that the JSP Pre-Touch tool is used as the mechanism for compiling the JSPs. The reason for this is that by using the JSP Pre-Touch tool to compile the JSPs, some additional code will be added to the compiled JSP classes to speed up the JSP Pre-Touch processing.

A secondary use is to class load ALL JSPs for an application into WebSphere Application Server. This removes the class load penalty paid by the first user to access a JSP file.

When not to use JSP Pre-Touch

Avoid running the Pre-Touch tool at application startup in production environments. Loading large numbers of infrequently used JSPs up front is not without tradeoffs. It can significantly increase the JVM heap size which can cause memory bottlenecks at startup and during garbage collection cycles. For instance, assume that an application has 20,000 JSP files. Assume that most customers, in normal running production environments, use about 2,000 of those JSPs for their job. Using the Pre-Touch will class load all 20,000 JSP files into WebSphere Application Server. As a result, running the JSP Pre-Touch during every application startup is neither practical nor very beneficial in this scenario. It may be desirable to run JSP Pre-Touch only once to prime the byte code cache. Or depending on the customer's situation, it may simply make more sense to let the byte code verification happen on a per-JSP basis as each JSP is invoked.

JSP Compilation

To enable the JSP Pre-Touch, there are three possible attributes that can be configured. Each of the attribute settings reside in the ibm-web-ext.xmi file of a web module (found in the WEB-INF directory). Setting the parameters in this file can be achieved using the WebSphere Development Studio Tools, the Application Assembly Tool, or any text editor.

The two most important attributes are prepareJSPs and prepareJSP Attribute. The third attribute, prepareJSPThreadCount, can be used on multi-processor systems.

prepareJSPs

When the attribute, prepareJSP, is present, all JSP files will be compiled on WebSphere Application Server startup. This process happens in a separate thread, so the WebSphere Application Server may, depending on the quantity and size of JSP that the application has, finish starting up before the JSP files are finished compiling. If starting and running the Web-enabled application seems slow, it could be because the JSP files are still compiling. Check the server log files to know when all of the JSP files have been compiled before doing any timings.

The numeric attribute value represents the minimum size (in kilobytes) that a JSP must be in order to also be class loaded and JIT-compiled. The default is 0, which causes all JSPs to be class loaded and JIT-compiled.

prepareJSPAttribute

Set the prepareJSP attribute to a value that is a request parameter composed of an alphanumeric that the JSP never expects to otherwise receive. This attribute is used to perform a quick exit from the service method of each JSP when they are prepared by this tool. This enables the tool to work much faster and prevent exceptions from showing in the WebSphere Application Server logs as a result of executing the JSP at startup.

If this attribute is not set on a Web-enabled application but the prepareJSP attribute is set, exceptions will be thrown by the application when the service method for the JSP is called because the WebFacing runtime has not been initialized. This will put many error messages in the log, use heap space, and cause the garbage collector to run. Here is a good rule of thumb — if precompiling the JSP, use the prepareJSP attribute setting as well.

prepareJSPThreadCount

Set the prepareJSPThreadCount numeric attribute to the number of threads that the user would like the JSP Pre-Touch tool to run in when processing the JSPs. Since a single thread makes use of just one processor, multi-processor systems may better utilize the JSP Pre-Touch tool by specifying a value equal to the number of processors on the system. The default setting for this attribute is 1, representing the number of threads that are created to perform pre-touch processing for this Web module.

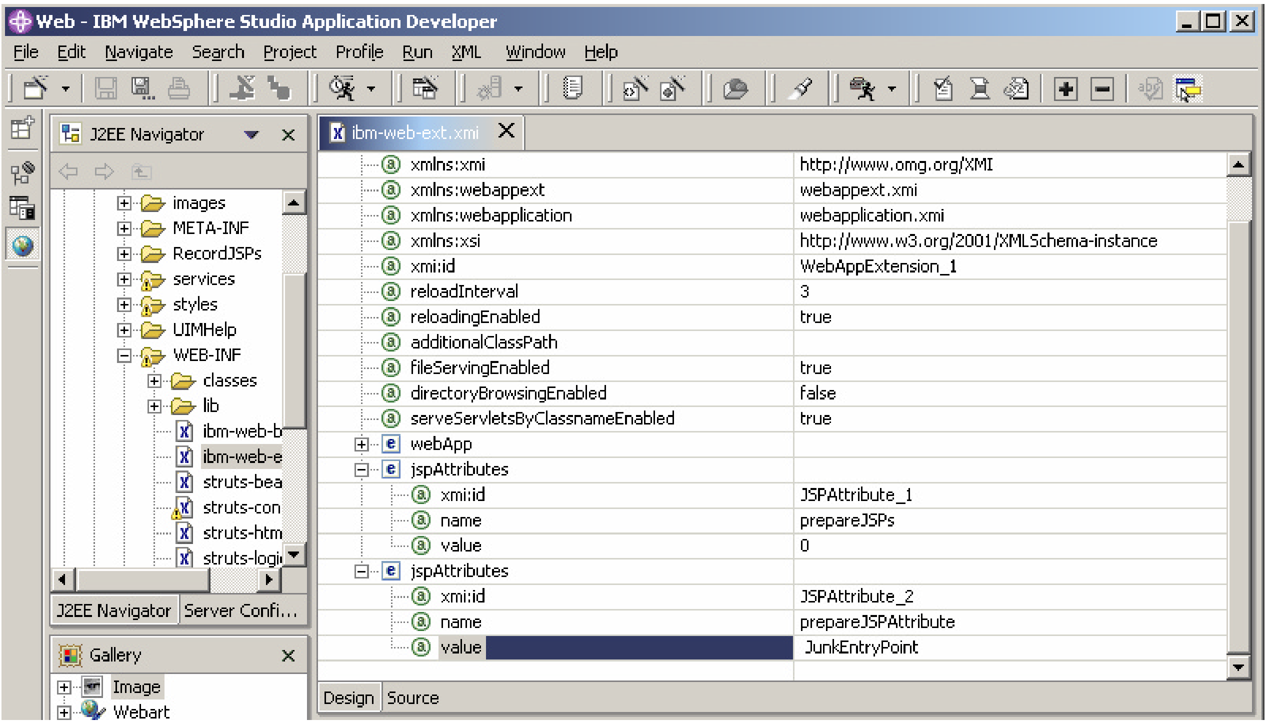
Configuring the Pre-Touch Attributes

The Pre-Touch attributes can be configured in three different ways.

? Using the WebSphere Development Studio Client
? Using a text editor to edit the deployed ibm-web-ext.xmi file

? Using the Application Assembly Tool shipped with WebSphere Application Server Base V5 (see product documentation).

Figure 4 shows an example of what the WebSphere Development Studio Client with the prepare attributes set from the WebFacing Tool looks like.



xmlns:xmi	http://www.omg.org/XMI
xmlns:webappext	webappext.xmi
xmlns:webapplication	webapplication.xmi
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xmi:id	WebAppExtension_1
reloadInterval	3
reloadingEnabled	true
additionalClassPath	
fileServingEnabled	true
directoryBrowsingEnabled	false
serveServletsByClassnameEnabled	true
webApp	
jspAttributes	
xmi:id	JSPAttribute_1
name	prepareJSPs
value	0
jspAttributes	
xmi:id	JSPAttribute_2
name	prepareJSPAttribute
value	JunkEntryPoint

Figure 4 — WebSphere Development Studio Client V5 with the prepare attributes set for the WebFacing Tool

It is possible to add or remove the JSP Pre-Touch attributes in the ibm-web-ext.xmi file. Be sure to modify the correct ibm-web-ext.xmi file if making changes to a deployed application. **CAUTION**: The ibm-web-ext.xmi file for a deployed application resides in two places — under the deployed application and under the cells directory when using a WebSphere Application Server V5.x. When modifying the file manually, make sure to modify the file under the cells directory path for the deployed application. For example:

> \QIBM\UserData\WebASE\ASE5\MYINSTNTANCE\config\cells\MYSYSTEM_MYINSTANCE\applications\ibmorder.ear\deployments\ibmorder\ibmorder.war\WEB-INF\ibm-web-ext.xmi

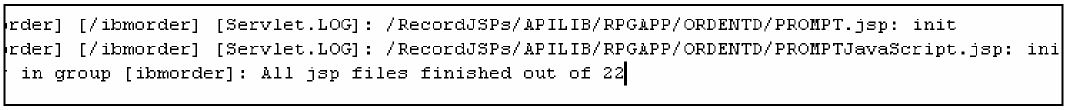
would be the location of the deployed location of the XMI file where attributes could be added or removed for WebSphere Application Server - Express V5.

Figure 5 shows the resulting ibm-web-ext.xmi file settings created from the WebSphere Development Studio Client Tools to edit the ibm-web-ext.xmi file as shown in Figure 4.0

```
<webappext:WebAppExtension xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:webappext="webappext.xmi" xmlns:webapplication="webapplication.xmi"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmi:id="WebAppExtension_1" reloadInterval="5"
reloadingEnabled="true" defaultErrorPage="error.jsp" additionalClassPath="" fileServingEnabled="false"
directoryBrowsingEnabled="false" serveServletsByClassnameEnabled="true">
  <webApp href="WEB-INF/web.xml#WebApp_ID"/>
  <jspAttributes xmi:id="JSPAttribute_1" name="prepareJSPs" value="0"/>
  <jspAttributes xmi:id="JSPAttribute_2" name="prepareJSPAttribute" value="JunkEntryPoint"/>
</webappext:WebAppExtensions>
```

Figure 5 — Displaying the contents of ibm-web-ext.xmi file highlighting the JSP attribute settings

During compilation, the JSP Pre-Touch tool writes information to the SystemOut.log file indicating progress was made. When all JSPs have finished compilation, a final message indicating that ALL JSP files have been compiled will exist in the log. Figure 6 (below) is a sample from the SystemOut.log file showing that in this example, ALL JSP files have finished out of 22.



```
rder] [/ibmorder] [Servlet.LOG]: /RecordJSPs/APILIB/RPGAPP/ORDENTD/PROMPT.jsp: init
rder] [/ibmorder] [Servlet.LOG]: /RecordJSPs/APILIB/RPGAPP/ORDENTD/PROMPTJavaScript.jsp: ini
 in group [ibmorder]: All jsp files finished out of 22
```

Figure 6 — Segment of SystemOut.log file from WebSphere Application Server - Express V5 showing “all JSP files finished” message

Once all JSP files have been compiled, these attributes can, and in many cases should, be removed from the deployed application. This will improve overall system performance upon subsequent server restarts since the WebSphere Application Server will skip the step of checking all JSP files for recompilation.

ByteCode Cache File

The ByteCode cache file is only available on an iSeries server. The cache file is an empty jar file that contains Java programs that can improve the startup performance of classes loaded by user class loaders. This is done by allowing the Java Program objects (JVAPGMs) created by user classloaders to be cached for reuse, avoiding JVAPGM creation and bytecode verification during the initial class load. WebSphere components (servlets and JSPs) are loaded by user classloaders and can take advantage of this feature.

When to Use ByteCode Caching

Consider the techniques described below if the system is experiencing performance problems in the following areas:

? Application server startup and termination
? Component runtime during first-touch of a JSP

The user classloader cache improves performance in two ways:

? Avoiding bytecode verification — If the program is already in the cache, bytecode verification is not performed again.
? Avoiding temporary creation of JVAPGMs — If the program is already in the cache, the existing JVAPGM is used. Since any optimization level can be stored in the cache, it is more practical to consider higher optimization levels.

In both cases, the first time the class is loaded (for example, before the cache is primed or after a class is changed), these functions are performed and the load is slower. A key point worth noting is that the byte code verification is permanent when using the cache file. Once the class is already in the cache, subsequent loads are much quicker.

Using the User Classloader Cache

To enable the user classloader cache, there is one Java System property that must be set, os400.define.class.cache.file. This setting is changed in the Custom Properties panel of the WebSphere Application Server administrative console.

os400.define.class.cache

The os400.define.class.cache setting specifies the full path name of a valid JAR file that holds the Java Program objects. This JAR file must contain a valid JAR entry. There are two ways to create a valid cache JAR file:

? Copy the /QIBM/ProdData/Java400/QDefineClassCache.jar file to an IFS directory and rename as desired

? Create a new cache JAR file as follows:

    o Start Qshell by entering the command STRQSH.

    o Switch to the directory where the JAR file will reside. Make the directory first, if necessary.

    o `mkdir /cache`

    o `cd /cache`

    o Create a dummy file to place in the JAR. The name can be anything, this example uses example.

    o `touch example`

    o Build the JAR file. This example names the JAR file `MyAppCache.jar`

    o `jar -cf MyAppCache.jar example`

    o Cleanup the dummy file

    o `rm example`

Additional User Class Loader Cache Settings

The additional class loader cache settings are optional and only valid if the cache file setting is used. For many customers, the default values are adequate and setting these parameters are unnecessary. The properties are:

? os400.define.class.cache.hours

? os400.define.class.cache.maxpgms

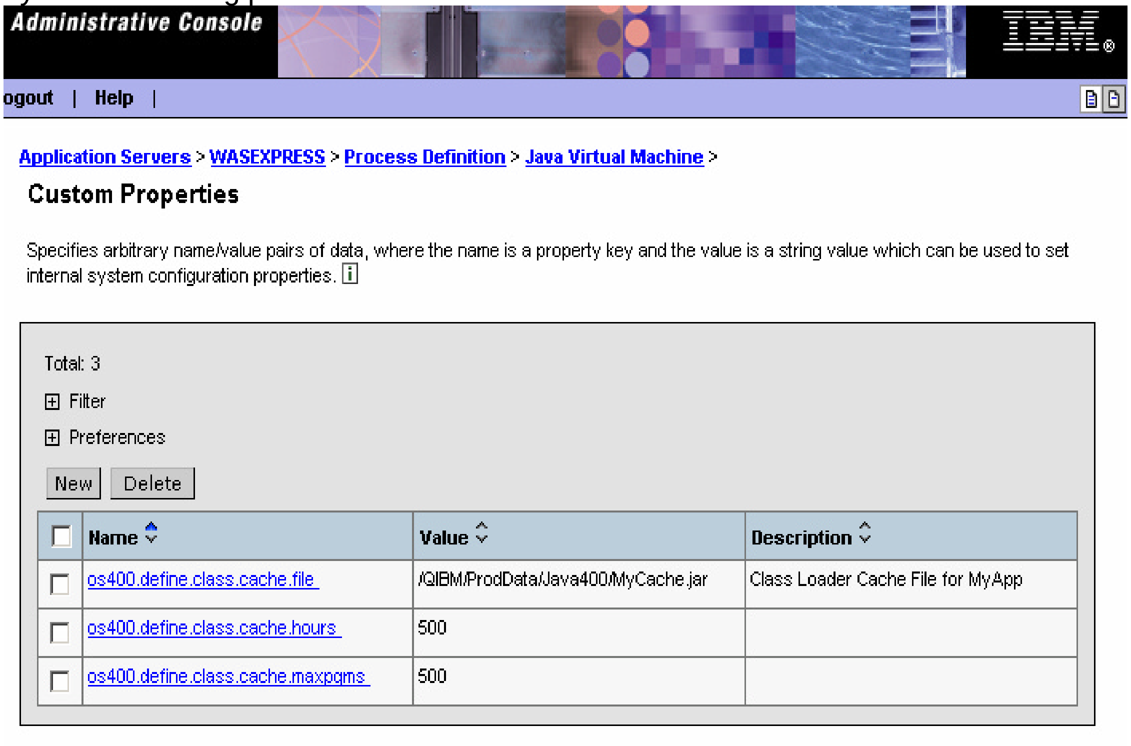
os400.define.class.cache.hours

The os400.define.class.cache.hours JVM setting specifies how long (in hours) an unused JVAPGM persists in the cache. When a JVAPGM has not been used and this timeout is reached, the JVAPGM is removed from the cache. The default value is 168 hours (one week). The maximum value is 9999 hours (about 59 weeks).

os400.define.class.cache.maxpgms

The os400.define.class.cache.maxpgms JVM setting specifies the maximum number of JVAPGMs the cache can hold. If this value is reached, the least recently used JVAPGMs is replaced first. The default value is 5000. The maximum value is 40000.

Figure 7 is an example of what a WebSphere Application Server 5.0 console with byte code caching parameters set looks like.


Administrative Console
IBM
ogout | Help |
Application Servers > WASEXPRESS > Process Definition > Java Virtual Machine >
Filter
Preferences
New
Delete

Custom Properties

Specifies arbitrary name/value pairs of data, where the name is a property key and the value is a string value which can be used to set internal system configuration properties.

Total: 3

□	Name	Value	Description
□	os400.define.class.cache.file	/QIBM/ProdData/Java400/MyCache.jar	Class Loader Cache File for MyApp
□	os400.define.class.cache.hours	500	
□	os400.define.class.cache.maxpgms	500	

Figure 7 — WebSphere Application Server 5.0 console byte code cache parameters

Figure 8 is a high- level view of how the JSP Pre-Touch tool works with or without the byte code caching option.

In the compile step, the JSP compiler determines if compilation is required. If the compiled JSP does not exist or is out of date with the class file, the JSP compiler will compile the JSP, saving the class file to disk.

The Class Load/Verification step loads the class file into memory. If the class is loaded from the Java class file, it must be byte code verified. If caching is enabled and the class can be loaded from the cache, the verification step is skipped. If caching is enabled and the file in the cache is out of date or does not exist, the class file is then written to the cache for future use.

The process is then repeated for the next JSP in the application until all JSPs are processed.

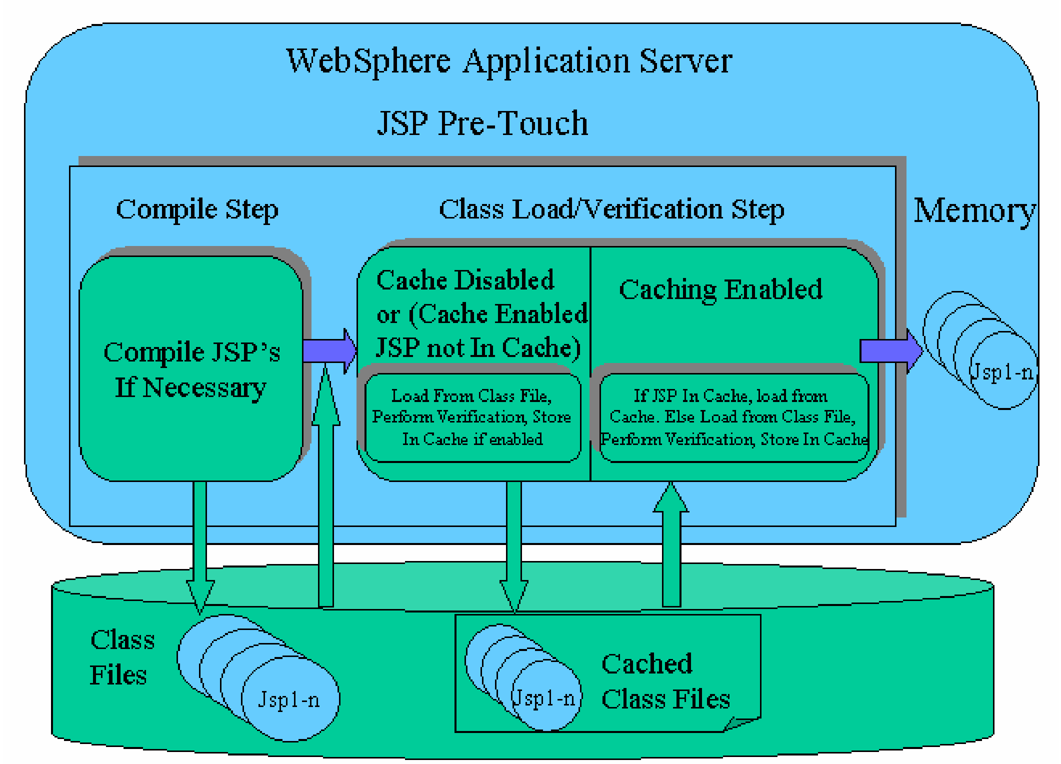

WebSphere Application Server
JSP Pre-Touch
Compile Step
Class Load/Verification Step
Memory
Compile JSP's If Necessary
Cache Disabled or (Cache Enabled JSP not In Cache)
Caching Enabled
Load From Class File, Perform Verification, Store In Cache if enabled
If JSP In Cache, load from Cache. Else Load from Class File, Perform Verification, Store In Cache
Jsp1-n
Class Files
Jsp1-n
Jsp1-n
Cached Class Files

Figure 8 —High-level view of how the JSP Pre-Touch tool works with or without the bytecode caching option

When not to use ByteCode Caching

The downside to using the ByteCode cache is that the cache file itself consumes disk resources and that there is no way to partially prime the cache. The choices are to prime the cache while running the application or configuring the JSP Pre-Touch tool, restarting the application server, and priming the cache with all class files for the application.

Priming the cache while running the application may not be a good option on low-end hardware models because the response times when priming the cache on first touch may be unacceptable to the customer.

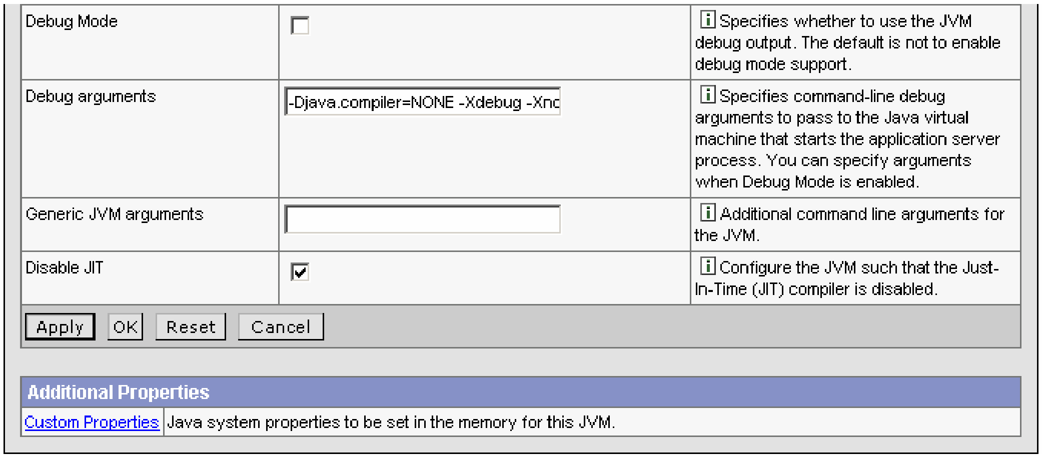
It is very difficult to make a blanket statement to use or not use the cache file. Most applications experience a slight gain (1.25 seconds) using the cache file the first time a JSP is touched versus not using the cache file.

Before using bytecode caching, test the application with precompiled JSPs only. If the first touch of the JSPs within an application are acceptable, using the ByteCode Cache file may not be necessary.

Just-In-Time compilation

Just-in-Time Compilation (JIT) is a method used by the Java Virtual Machine to speed up the execution of Java programs. Testing has shown that turning on the JIT compiler for a Web-enabled application does improve performance and it provides about 10% better performance as the direct execution environment on iSeries. Users who enable the JIT compiler can expect a performance hit of approximately two seconds when accessing their application for the first time. This is due to the JIT compiling WFRun.jar, (the WebFacing Runtime classes). The default setting for JIT is enabled. Users who do not wish to experience the JIT compilation penalty at first should disable JIT compilation. Disabling the jit Compiler is done in the WebSphere Application Server Console.

Figure 9 is an example of what a WebSphere Application Server V5.0 console with JIT Disabled.



Debug Mode	□	Specifies whether to use the JVM debug output. The default is not to enable debug mode support.
Debug arguments	-Djava.compiler=NONE -Xdebug -Xnc	Specifies command-line debug arguments to pass to the Java virtual machine that starts the application server process. You can specify arguments when Debug Mode is enabled.
Generic JVM arguments		Additional command line arguments for the JVM.
Disable JIT	☑	Configure the JVM such that the Just-In-Time (JIT) compiler is disabled.

Additional Properties	
Custom Properties	Java system properties to be set in the memory for this JVM.

Figure 9 — WebSphere Application Server Version 5.0 Console with JIT Compiler Disabled

Additional Performance Enhancements

Version 5.0 of the IBM WebFacing Tool

There have been a significant number of enhancements delivered with V5.0 of the IBM WebFacing Tool including:

? Support for viewing and printing spooled files (WebSphere Development Studio Client V5 Advanced Edition Only)

?????Struts-compliant code generated by the IBM WebFacing Tool conversion process which sets the foundation for extending the IBM Webfaced applications using struts-compliant action architecture (WebSphere Development Studio Client Advanced Edition Only)

? Automatic configuration for UTF-8 support when deployed to WebSphere Application Server V5.0

? Support for function keys within window records

? Enhanced hyperlink support

? Support to enable compression to improve response times on slow connections

? Improved memory optimization for record I/O processing

The two important enhancements from a performance perspective will be discussed below. For other information related to the IBM WebFacing Tool V5.0, please refer to the following Web site:
ibm.com/software/awdtools/wdt400/about/webfacing.html

Compression

WebFacing has been enhanced to support compression in version 5.0 of the IBM WebFacing Tool.

Display File Record I/O Processing

Display file record I/O processing has been optimized to decrease the WebSphere Application Server runtime memory utilization. This has been accomplished by enhancing the Webfacing runtime to better utilize the Java objects required for processing display I/O requests for each end user transaction. Formerly on each record I/O, Webfacing had to create a record data bean object to describe the I/O request, and then create the record bean using this definition to pass the I/O data to the associated JSP. These definition objects were not reused and were created for each user. With the optimization implemented in V5.0, the record bean definitions are now reused and cached so that one instance for each display file record can be shared by all users.

This optimization has decreased the overall memory requirements for the IBM WebFacing Tool V5.0 versus V4.0. This memory savings helps reduce the total memory required by the WebSphere Application Server, which is referred to as the JVM Heap Size. The amount of memory savings depends on a number of parameters, such as the complexity of the screens (based on number of fields per screen), the transaction rate, and the number of concurrent end users. On measurements made with approximately 250 users and varying screen complexity, the JVM Heap decreased by approximately 5% for simple to moderate screens (99 fields per screen) and up to 20% for applications with more complex screens (600 fields per screen). When looking at the overall memory requirements for an application, the JVM Heap size is just one component. When running the back-end application on the same server as the WebSphere Application server, the overall decrease in system memory required for the Webfaced application will be less.

In terms of CPU utilization, this optimization has provided some benefit by a decrease of up to 10% less for complex workloads. However, when taking into account the overall CPU utilization for a Webfaced application (Webfacing plus the application), expect equal or slightly better performance with the WebFacing Tool V5.0.

Tuning the Record Definition Cache

In order to best use the optimization provided by this enhancement, servlet utilities have been included in the Webfacing support to assess cache efficiency, set the cache size, and preload it with the most frequently accessed record definitions. If the Record Definition Cache is not used or tuned improperly, there will be degraded performance of the IBM WebFacing Tool V5.0 versus V4.0.

When set to an appropriate level for the Webfaced application, the Record Definition Cache can provide a decrease in memory usage, and slightly decreased processor usage. The number of record definitions that the cache will retain is set by an initialization parameter in the Webfaced application's deployment descriptor (web.xml). By changing the cache size, the Webfaced application can be tuned for best performance and minimum memory requirements. The cache size determines the number of record data definitions that will be retained in the cache. There is one record data definition for each record format.

Cache Size	Effect
Too small	When the cache size is set too small for the Webfaced application, it will adversely affect the performance. In this case, the definitions would be cached then discarded before being re-used. There is significant overhead to create the record definitions.
Correct	With the cache set correctly, 90% of all accessed record data definitions would be retained in the cache with few cache misses for not commonly used records.
Too large	If the cache is set too large then all record data definitions for the Webfaced application would be cached likely consuming memory for seldom used definitions.

In order to determine what the correct size for a given Webfaced application, the number of commonly used record formats needs to be estimated. This can be used as a starting point for setting the cache size. The default size, if no size is specified, would be 600 record data definitions. To set the cache size to something other than the default size, add a session context parameter in the

Webfaced applications web.xml file. In the following example, the cache size is set to 200 elements, which may be appropriate for a very small application, like the Order Entry example program.

```
<context-param>
  <param-name>WFBeanCacheSize</param-name>
  <param-value>200</param-value>
  <description>WebFacing Record Definition Bean Cache Size</description>
</context-param>
```

NOTE: For information on defining a session context parameter in the web.xml file, refer to the WebSphere Application Server Information Center. It is also possible to edit the web.xml file of a deployed application. Typically, this file will be located in the following directory for WebSphere Application Server V5.0 applications:

/QIBM/UserData/WebAS5/Base/<application-server>/config/cells/..../WEB_INF

It can also be found in the following directory for WebSphere Application Server - Express V5.0 applications:

/QIBM/UserData/WebASE/ASE5/<application-server>/config/cells/..../WEB_INF

Cache Management — Definition Cache Content Viewer

To assist with managing the Record Definition Cache, two servlets can be enabled. One is used to display the elements currently in the cache and the other can be used to load the cache. Both of these servlets are not normally enabled in a WebFacing application in order to prevent misuse or exposure of data.

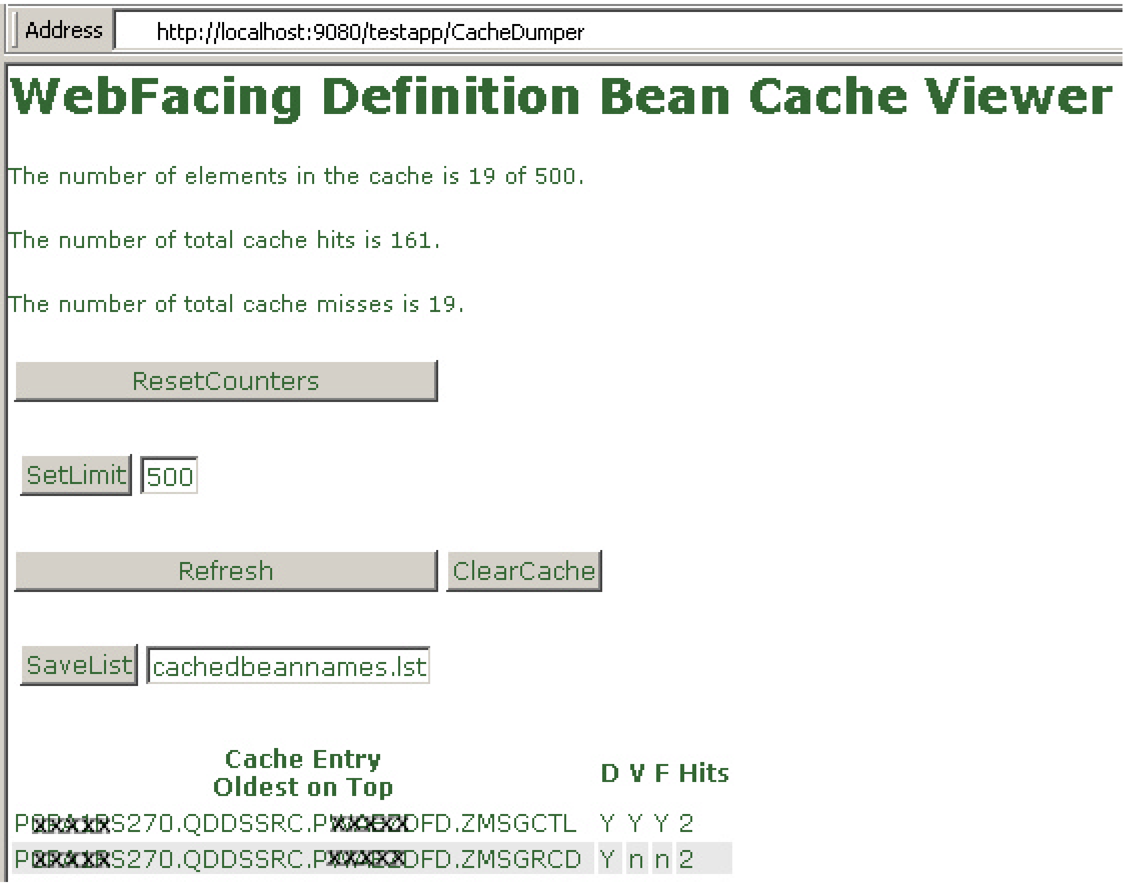
To enable the servlet that will display the contents of the cache, first add the following segments to the Webfaced application's web.xml:

```
<servlet>
  <servlet-name>CacheDumper</servlet-name>
  <display-name>CacheDumper</display-name>
  <servlet-class>com.ibm.etools.iseries.webfacing.diags.CacheDumper</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>CacheDumper</servlet-name>
  <url-pattern>/CacheDumper</url-pattern>
</servlet-mapping>
```

This servlet can then be invoked with a URL like:
http://<server>:<port>/<webapp>/CacheDumper.

Then, a Web page like that shown below will be displayed. Notice that the total number of cache hits and misses are displayed, as are the hits for each record definition.



http://localhost:9080/testapp/CacheDumper

WebFacing Definition Bean Cache Viewer

The number of elements in the cache is 19 of 500.

The number of total cache hits is 161.

The number of total cache misses is 19.

SetLimit: 500

SaveList: cachedbeannames.lst

Cache Entry Oldest on Top	D	V	F	Hits
P[redacted]S270.QDDSSRC.P[redacted]DFD.ZMSGCTL	Y	Y	Y	2
P[redacted]S270.QDDSSRC.P[redacted]DFD.ZMSGRCD	Y	n	n	2

Figure 10 — Web page displaying the contents of the cache

Refer to the following table for the functionality provided by the Cache Viewer servlet. NOTE: The D V F column heading stand for Data, Feedback, View and have no significance related to performance.

Cache Viewer Button operations	
Button	**Operation**
Reset Counters	Resets the cache hit-and-miss counters back to 0
Set Limit	Temporarily sets the cache limit to a new value. Setting the value lower than the current value will cause the cache to be cleared as well.
Refresh	Refresh the display of cache elements
Clear Cache	Drop all the cached definitions
Save List	Save a list of all the cached record data definitions. This list is saved in the RecordJSPs directory of the Webfaced application. The actual record definitions are not saved, just the list of what record definitions are cached. Once the cache is optimally tuned, this list can be used to preload the Record Definition cache.

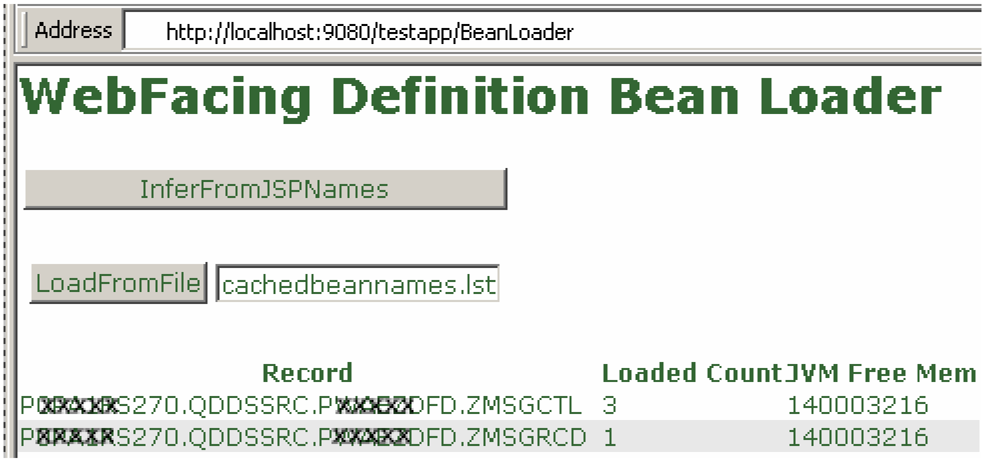
Cache Management — Record Definition Loader

As a companion to the Cache Content Viewer tool, there is also a Record Definition Cache Loader tool, which is often referred to as the Bean Loader. This servlet can be used to preload the cache to aid in the determination of the optimal cache size, and then finally, to preload the cache for production use. To enable this servlet, add the following two xml segments in the web.xml file.

```
<servlet>
  <servlet-name>BeanLoader</servlet-name>
  <display-name>BeanLoader</display-name>
  <servlet-class>com.ibm.etools.iseries.webfacing.diags.BeanLoader</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>BeanLoader</servlet-name>
  <url-pattern>/BeanLoader</url-pattern>
</servlet-mapping>
```

Invoking this servlet will present a Web page similar to the following:



Address http://localhost:9080/testapp/BeanLoader

WebFacing Definition Bean Loader

Record	Loaded Count	JVM Free Mem
P[redacted]S270.QDDSSRC.P[redacted]DFD.ZMSGCTL	3	140003216
P[redacted]S270.QDDSSRC.P[redacted]DFD.ZMSGRCD	1	140003216

Figure 11 — Record Definition Cache Loader tool

Refer to the following table for the functionality provided by the Record Definition Loader servlet.

Record Definition Loader Button operations	
Button	**Operation**
Infer from JSP Names	Will cause the loader servlet to infer record definition names from the names or the JSPs contained in the RecordJSPs directory. It will not find all the record definitions but it will get most of them.
Load from File	Will load the record definitions listed in a file in the RecordJSPs directory. Typically, this file is created with the CacheDumper servlet previously described.

The Record Definition Loader servlet can also be used to preload the bean definitions when the Webfaced application is started. To enable this, the servlet definition in the web.xml needs to be updated to define two init parameters: FileName and DisableUI. The FileName parameter indicates the name of the file in the RecordJSPs directory that contains the list of definitions to preload the cache with. The DisableUI parameter indicates that the Web UI (as presented above) would be disabled so that the servlet can be used to safely preload the definitions without exposing the Webfaced application.

```
<servlet>
  <servlet-name>BeanLoader</servlet-name>
  <display-name>BeanLoader</display-name>
  <servlet-
class>com.ibm.etools.iseries.webfacing.diags.BeanLoader</servlet-
class>
  <init-param>
    <param-name>FileName</param-name>
    <param-value>cachedbeannames.lst</param-value>
  </init-param>
  <init-param>
    <param-name>DisableUI</param-name>
    <param-value>true</param-value>
  </init-param>
  <load-on-startup>10</load-on-startup>
  </servlet>
```

WSADMIN and WebSphere Application Server Scripts

The release of WebSphere Application Server - Express V5.0 for iSeries brought a new age to usability with the Apache Integrated GUI. This GUI is an easy way for customers to create, start, stop, and install WebSphere Application Server applications. The reviews for ease-of-use have been outstanding and many users find this interface superior in many ways.

The caveat to using the Integrated GUI for creating, starting, stopping, and installing applications is that it too consumes system resources. The Integrated GUI runs in a separate application server, running TomCat. On a system with limited memory, having two application servers running at the same time may cause some significant performance problems.

Customers trying to run and manage applications within the WebSphere Application Server environment on a minimum configuration of 512MB of memory should consider an alternative to the Integrated GUI. This can be done using the scripting support provided by WSADMIN.

More information about WSADMIN and the scripts can be found in InfoCenter at: http://publib.boulder.ibm.com/iseries/v5r2/ic2924/index.htm?info/rzamy/50/express.htm

WSADMIN

WebSphere Application Server - Express V5.0 provides a command line administrative tool named wsadmin, which can be used to run administrative commands interactively or through the use of Jacl script files. The wsadmin tool uses the Bean Scripting Framework (BSF), which supports a variety of scripting languages to configure and control WebSphere Application Server - Express. In WebSphere Application Server - Express, wsadmin supports only the Jacl scripting language.

The wsadmin launcher makes Java objects available through language specific interfaces. Scripts use these objects for application management, configuration, operational control, and communication with MBeans running in WebSphere Application Server processes.

WebSphere Application Server - Express System Management separates administrative functions into these categories:

? Configuration — Related to the configuration of WebSphere Application Server - Express installations
? Operation — Related to the currently running objects in WebSphere Application Server - Express installations.
? Application management — Related to installing, uninstalling, and managing enterprise applications.

To install an application using WSADMIN, first start a wsadmin session.

1. Enter the Start Qshell (STRQSH) command on an OS/400 command line.
2. Use the cd command to change to the bin directory of the product installation root:
   `cd /QIBM/ProdData/WebASE/ASE5/bin`
3. At the Qshell prompt, enter this command:
   `wsadmin` -instance *instance*
   where *instance* is the name of the instance to administer.

   In WebSphere Application Server - Express V5, this is the application server name specified when the server was created using the Integrated GUI. When using WebSphere Application Server V5, specify ‘default’ if using the default instance; otherwise, use the instance name specified using the crtwasinst command.

4. This command uses the EAR file and command option information to install the application
   `$AdminApp install *earfile* {-server *myAppSvr*}`
   where *earfile* is the fully qualified path of the EAR file that is installed and *myAppSvr* is the name of WebSphere Application Server instance.
5. Before exiting the interactive session, run the command below to save the configuration changes:
   $AdminConfig save

crtwasinst

WebSphere Application Server V5 (all version) supply an instance creating script called crtwasinst for creating a new Application Server Instance. It can be run from QSHELL. Executing the crtnewinst job on the QSHELL command line will actually run the script in the interactive memory pool. If that pool is starved for memory, running it probably will not help.

There are several different ways to force a Qshell command to run into pool 2 if needed. For instance:

```
SBMJOB CMD(STRQSH CMD('cd
/QIBM/ProdData/WebASE/ASE5/bin;crtwasinst –instance MyInstance)
```

runs the crtwasinst Qshell command as a batch job running in pool 2.

startServer, stopServer

The startServer and stopServer scripts can be used for starting and stopping WebSphere Application Server instances from Qshell. The command can be started similar as above to guarantee the memory pool it runs in.

Browser Caching

The most efficient way to retrieve statistical content is to have it cached at the browser. Microsoft Internet Explorer has four different setting for caching static content. The automatic setting within Internet Explorer is the recommended setting. The “automatic” cache setting means that when a page is visited for the first time, static content is cached. When the page is visited again within the same browser session, no request for the static content is made to the server — it is retrieved from the browser cache file instead. This results in fewer requests to the server and reduces network bandwidth. If a page is visited again from a different browser session, a request is made for the content. If the browser has the most recent copy of the content in cache, the server responses back with a very short "304-Not Changed" response instead of responding with the actual static content. In most circumstances, the 304 response is much smaller than the static content being requested.

IBM HTTP Server Powered By Apache to Serve Static Content

With the birth of J2EE, developers were introduced to WebArchive and Enterprise Application Resource files. These files are self contained entities that make it easy to distribute and install into a WebSphere Application Server.

Many developers install the .EAR or .WAR files into WebSphere application server and configure the HTTP server to forward all requests to the WebSphere Application Server. It is much faster to serve static HTML content with a

WebServer (Apache) than it is with a WebApplication Server (WebSphere Application Server).

When an IBM WebFaced application is generated, by default, the tool sets a flag “fileServingEnabled” in the ibm-web-ext.xmi file to true. This impacts the plugin-cfg.xml file and causes the HTTP plugin code to forward all requests to the WebSphere Application Server, including static content requests. This makes it easy to deploy the application to the Web Application Server, but may not yield optimal performance.

To configure the HTTP Server to serve up static content, the basic steps are as follows (additional information is included in the subsequent sections):

? Run the “Add a Directory to the Web” wizard from the HTTP Configuration
? Change permissions on the deployed application allowing QTMHTTPSVR read/execute access to static content
? Set fileServingEnabled = false for the deployed application
? Regen the WebSphere Plugin

Add a Directory to the Web

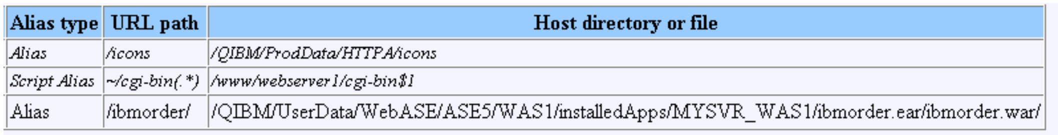
Adding a directory for the IBM HTTP Server Powered by Apache Web server is simple when using the built-in wizard. There are two key points to remember when configuring the HTTP Server to serve static content from an IBM WebFaced Application:

1. When prompted for which directory to serve from, select the directory ending in the .war extension. For example, if an application is installed within WebSphere Application Server - Express on a system called MYSYSTEM, the proper directory would be:

   /QIBM/UserData/WebASE/ASE5/WASEXPRESS/installedApps/MYSYSTEM_WASEXPRESS/ibmorder.ear/ibmorder.war/

2. When prompted for the directory alias, use the same context-root as the deployed application.

Figure 12 is an example of what the URL mappings may look like with an application deployed to WebSphere Application Server - Express V5.



Alias type	URL path	Host directory or file
Alias	*/icons*	*/QIBM/ProdData/HTTPA/icons*
Script Alias	*~/cgi-bin(.*)*	*/www/webserver1/cgi-bin$1*
Alias	/ibmorder/	/QIBM/UserData/WebASE/ASE5/WAS1/installedApps/MYSVR_WAS1/ibmorder.ear/ibmorder.war/

Figure 12 — Integrated GUI showing Alias pointing to WebSphere Application Server - Express V5 directory

This image shows the URL Mappings for a deployed application to WebSphere Application Server - Express V5.0. The url path, /ibmorder/ , is the context root of the deployed application. The path it aliases is the deployed path of the .war directory.

Change Permissions on the deployed application

The IBM HTTP Server Powered by Apache requires Read/Execute permissions to be able to serve static content. By default, the application installed into WebSphere Application Server will not have sufficient permissions to serve static content. This is evident when trying to serve static content and receiving a 403 Permission Error in the browser.

The solution to this problem is quite easy — grant QTMHTTPSVR Read/Execute permission to all static content of the deployed application. QTMHTTPSVR is the user id used by the IBM HTTP Server Powered by Apache.

The easiest way to grant these permissions is to use the recursive "chgaut", (Change Authority), command. The example below shows how to change all .gif files below the current directory to *RX for user QTMHHTTP.

```
find . -name '*.gif' -exec system "CHGAUT OBJ('{}') user(QTMHHTTP) DTAAUT(*RX)
OBJAUT(*NONE)" \;
```

Similar commands could be executed for other file types or for all files. For security purposes, it is recommended to grant user QTMHHTTP only files that the HTTP Server will be serving.

Set fileServingEnabled for the deployed application

When the IBM WebFaced application is generated, by default, the tool sets a flag "fileServingEnabled" in the ibm-web-ext.xmi file to true. This setting forces all requests of the context-root to be served by WebSphere Application Server.

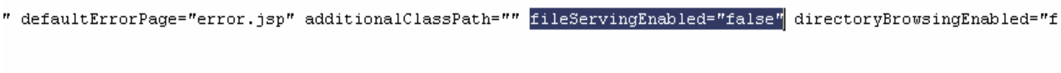
Solution developers using the HTTP Server to serve up static content will want to distribute the ibm-web-ext.xmi file with the fileServingEnabled setting set to false. There are two reasons for this

1. Editing deployed files can be risky for untrained users
2. Websphere Application Server V5.0 (Express, Base, Network Deployment) copy this file at installation time to the cells directory structure. Changing this file at runtime to the deployed application directory will have no effect. It must be changed in the proper cells directory structure and can be confusing to a new user.

It is possible to change the fileServingEnabled attribute in the ibm-web-ext.xmi file. Be sure to modify the correct ibm-web-ext.xmi file if making changes to a deployed application. **CAUTION**: The ibm-web-ext.xmi file for a deployed application resides in two places — under the deployed application and under the cells directory when using a WebSphere Application Server V5.x. When modifying the file manually, make sure to modify the file under the cells directory path for the deployed application. For example, the following would be the location of the deployed location of the XMI file where attributes could be added or removed for WebSphere Application Server - Express V5.

\QIBM\UserData\WebASE\ASE5\MYINSTNTANCE\config\cells\MYSYSTEM_MYINSTANCE\applications\ibmorder.ear\deployments\ibmorder\ibmorder.war\WEB-INF\ibm-web-ext.xmi

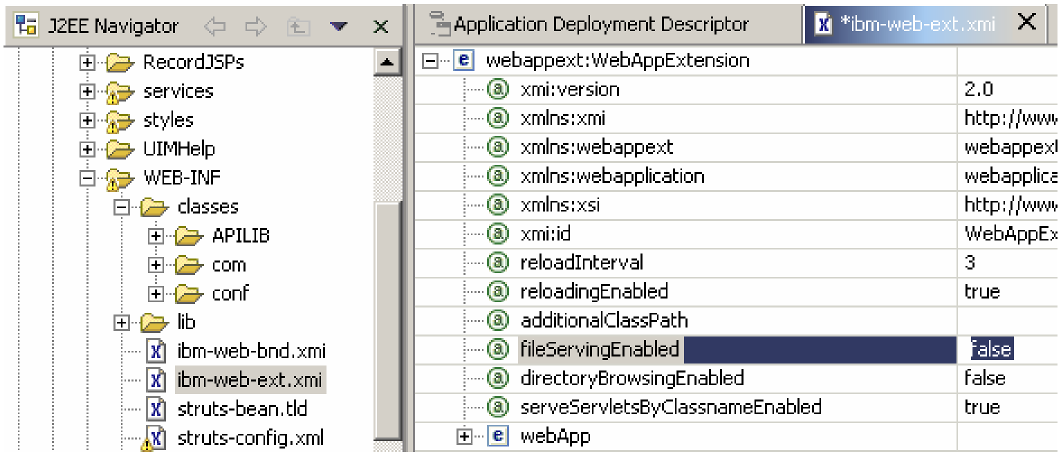
Figure 13 displays a line in the ibm-web-ext.xmi file highlighting the fileServingEnabled parameter.



```
" defaultErrorPage="error.jsp" additionalClassPath="" fileServingEnabled="false" directoryBrowsingEnabled="f
```

Figure 13 — fileServingEnabled parameter in ibm-web-ext.xmi file

This setting is very easy to change from the WebSphere Development Studio Client tools. Simply switch to the Web Perspective and open the project's \WEB-INF\lib\ibm-web-ext.xmi file. Change the fileServingEnabled attribute as needed. Figure 14 is a screen shot of the ibm-web-ext.xmi file opened in the Web Perspective of an IBM WebFaced Application. Notice the fileServingEnabled parameter and how easy it is to modify.



webappext:WebAppExtension	
xmi:version	2.0
xmlns:xmi	http://www
xmlns:webappext	webappext
xmlns:webapplication	webapplica
xmlns:xsi	http://www
xmi:id	WebAppEx
reloadInterval	3
reloadingEnabled	true
additionalClassPath	
fileServingEnabled	false
directoryBrowsingEnabled	false
serveServletsByClassnameEnabled	true
webApp	

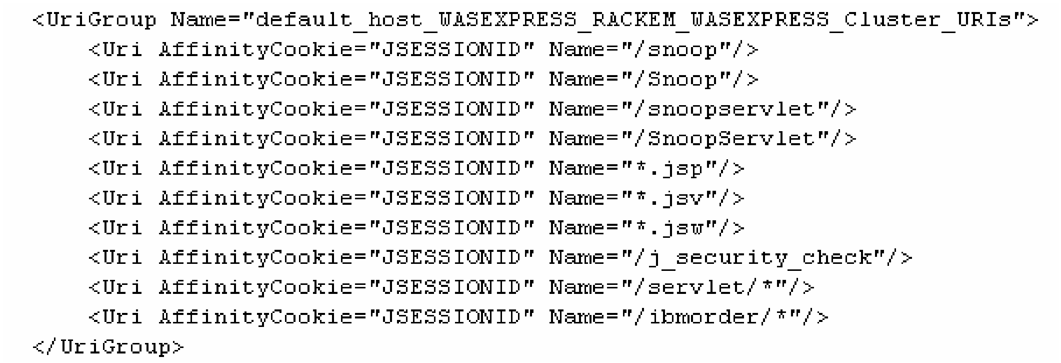
Figure 14 — Setting the fileServingEnabled parameter using the WebSphere Development Studio Client's Web Perspective

Regen the WebSphere Plugin

The WebSphere Application Server plugin file plugin-cfg.xml, is an XML file that is used by the application server plugin code to determine if the WebSphere Application Server is responsible for serving the content.

If the fileServingEnabler setting is set to true in the ibm-web-ext.xmi file, when the plugin file is generated, it will contain an entry for the Webfacing application that indicates that ALL content associated with the context root is to be served by WebSphere Application Server.

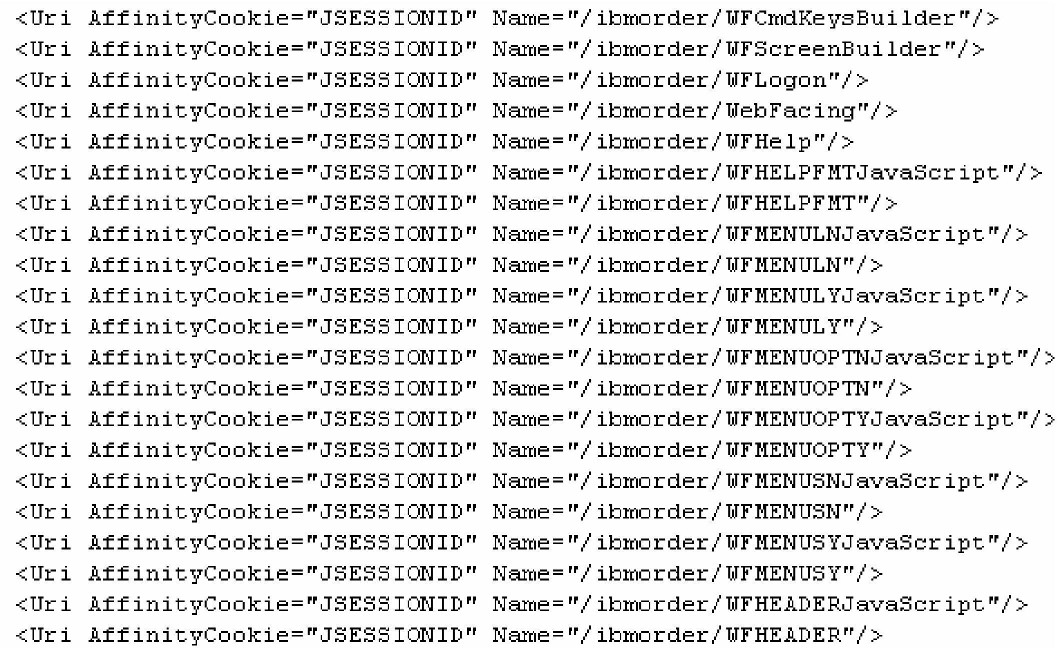
Figure 15 shows what the plugin-cfg.xml file contains when the fileServingEnabler value is set to true for a deployed application. Notice the entry /ibmorder/*. This tells the plugin code that WebSphere Application Server can handle all URIs containing /ibmorder. For all intensive purposes, all content.



```
<UriGroup Name="default_host_WASEXPRESS_RACKEM_WASEXPRESS_Cluster_URIs">
    <Uri AffinityCookie="JSESSIONID" Name="/snoop"/>
    <Uri AffinityCookie="JSESSIONID" Name="/Snoop"/>
    <Uri AffinityCookie="JSESSIONID" Name="/snoopservlet"/>
    <Uri AffinityCookie="JSESSIONID" Name="/SnoopServlet"/>
    <Uri AffinityCookie="JSESSIONID" Name="*.jsp"/>
    <Uri AffinityCookie="JSESSIONID" Name="*.jsv"/>
    <Uri AffinityCookie="JSESSIONID" Name="*.jsw"/>
    <Uri AffinityCookie="JSESSIONID" Name="/j_security_check"/>
    <Uri AffinityCookie="JSESSIONID" Name="/servlet/*"/>
    <Uri AffinityCookie="JSESSIONID" Name="/ibmorder/*"/>
</UriGroup>
```

Figure 15 — plugin-cfg.xml file for and IBM WebFaced application when fileServingEnabled = true

Changing the fileServingEnabler setting to false in the ibm-web-ext.xml file and then regenerating the plugin file will change the plugin to contain entries to dynamic content that can be served by WebSphere Application Server only. Figure 16 below shows what the plugin-cfg.xml file looks like after changing the fileServingEnabler value to false for the same application. Notice how the \ibmorder* has been replaced with specific entries and specific URIs for serving dynamic content.



```
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFCmdKeysBuilder"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFScreenBuilder"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFLogon"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WebFacing"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFHelp"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFHELPFMTJavaScript"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFHELPFMT"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFMENULNJavaScript"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFMENULN"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFMENULYJavaScript"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFMENULY"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFMENUOPTNJavaScript"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFMENUOPTN"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFMENUOPTYJavaScript"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFMENUOPTY"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFMENUSNJavaScript"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFMENUSN"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFMENUSYJavaScript"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFMENUSY"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFHEADERJavaScript"/>
<Uri AffinityCookie="JSESSIONID" Name="/ibmorder/WFHEADER"/>
```

Figure 16 — plugin-cfg.xml file for an IBM WebFaced Application when fileServingEnabled = false for the same IBM WebFaced application

An excellent article in the WebSphere Developers Journal entitled “Handling Static Content in WebSphere Application Server” provides details on the performance gains by using the IBM HTTP Server Powered by Apache to serve static content. This article can be found at:
ibm.com/software/wsdd/techjournal/0211_brown/brown.html

Application Deployment Options

When discussing deployment options of Web enabled applications, think about both new deployments and updates to running environments. New deployments are just that new. Think of new deployments as installing an .EAR file containing all of the JSPs, servlets, and compiled JSPs that make up an application to a customer for the first time.

Updated deployments are updates to a running production environment. Consider the situation where a customer has installed an application and has been running in a production environment for some time. Since then either the customer or solution developer may have updated the application. Possibly the DDS source has changed or additional customization has been added. What procedure works best for providing a smooth transition to the new application code in a production environment?

Deploying a new application

Earlier in this paper, it was demonstrated that precompiling JSPs significantly improves the first touch page response time of a JSP. It is possible to ship precompiled JSP files with an .EAR or .WAR deployment file so that the overhead of recompiling is not necessary at a customer site. The steps necessary to package up an application with precompiled JSPs are as follows:

1. Precompile all JSPs on a development machine using either the JSP Batch or JSP Pre-Touch mechanism
2. Copy all of the compiled JSP files from the temp directory to the deployed application directory
3. Use the EARExpander tool repackage .EAR file

Precompile all JSPs on development server

As described earlier, there are a couple of techniques to precompile all of the JSP files. Whatever technique the developer chooses, the result is that all compiled JSP files end up residing in the temp directory of the deployed application. For WebSphere Application Server - Express that directory would look similar to this: \QIBM\UserData\WebASE\ASE5\MYINSTNTANCE\Temp\MYSYSTEM_MYINSTANCE\MYINSTANCE\MYAPPLICATOIN.ear\MYAPPLICATION.war

(Substitute MYSYSTEM, MYINSTANCE, and MYAPPLICATION for actual system, instance, and application names.)

Copy the Compiled JSPs

The next step in the process is to copy the compiled JSP directories to the deployed application directories. This can be accomplished in several ways. A simple way is to map a network drive on a PC to the IFS directory structure and drag-and-drop directories.

Figure 17 shows the directory structure for an application called ibmorder.ear deployed to WebSphere Application Server - Express V5. The full path is: \QIBM\UserData\WebASE\ASE\MYINSTANCE\installedApps.

Below the installedApps directory is a directory consisting of the MYSYSTEM_MYINSTANCE, then all of the .ear directories for the installed applications. Below the ibmorder.ear directory is a .war directory and a series of directories below that. Notice in the WEB-INF directory, there are a series of directories containing the WebFaced JSPs and Java Servlet code.

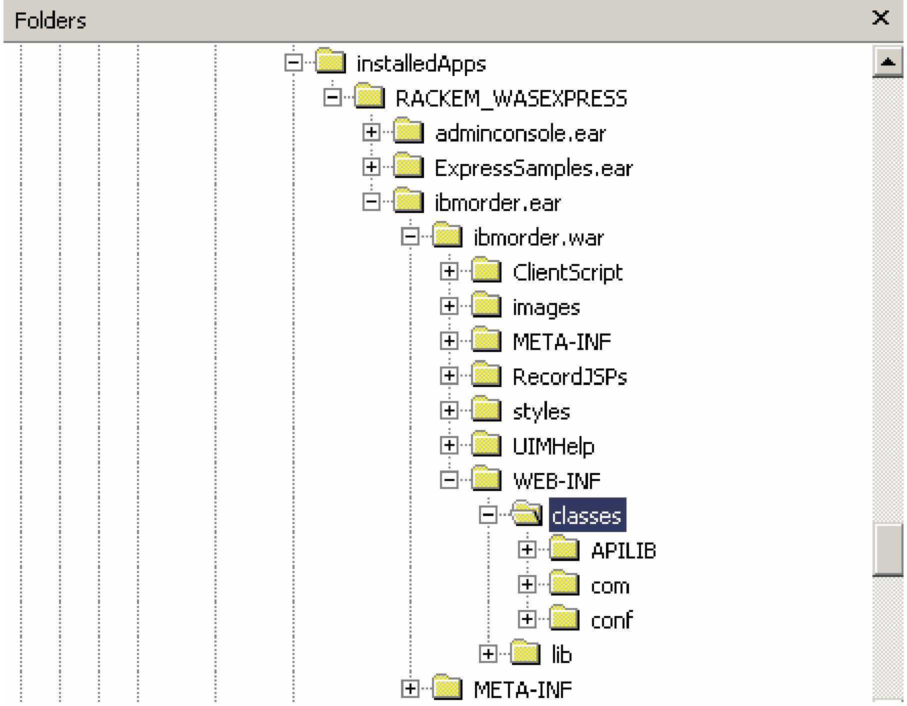


Figure 17 — Directory Structure of Deployed Application without precompiled JSPs

Figure 18 shows the directory structure under the temp directory where all of the precompiled JSPs have been compiled. The full path is:
\QIBM\UserData\WebASE\ASE\MYINSTANCE\temp.
Below the temp directory is a directory consisting of the MYSYSTEM_MYINSTANCE\MYINSTANCE and then all of the directories for the installed applications without the .ear extensions. Below the ibmorder directory is a MYINSTANCE.war directory and directories below that. All of the directories below the MYINSTANCE.war directory should be copied to the classes directory for the deployed application:
\QIBM\UserData\WebASE\ASE\MYINSTANCE\installedApps\MYSERVER_MYINSTANCE\MYAPP.ear\MYAPP.war\WEB-INF\classes

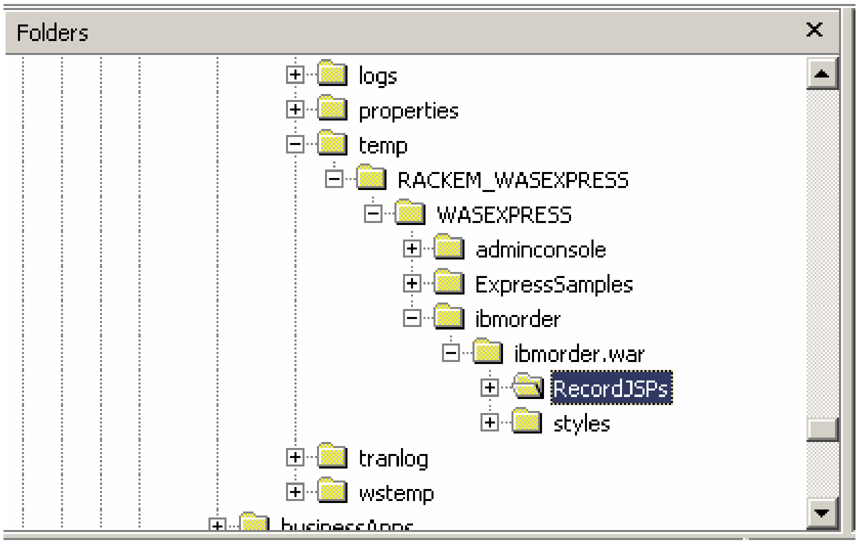


Figure 18 — Directory Structure of precompiled JSPs without precompiled JSPs

Figure 19 displays the results of copying the RecordJSPs and styles directories to the classes directory of the deployed application.

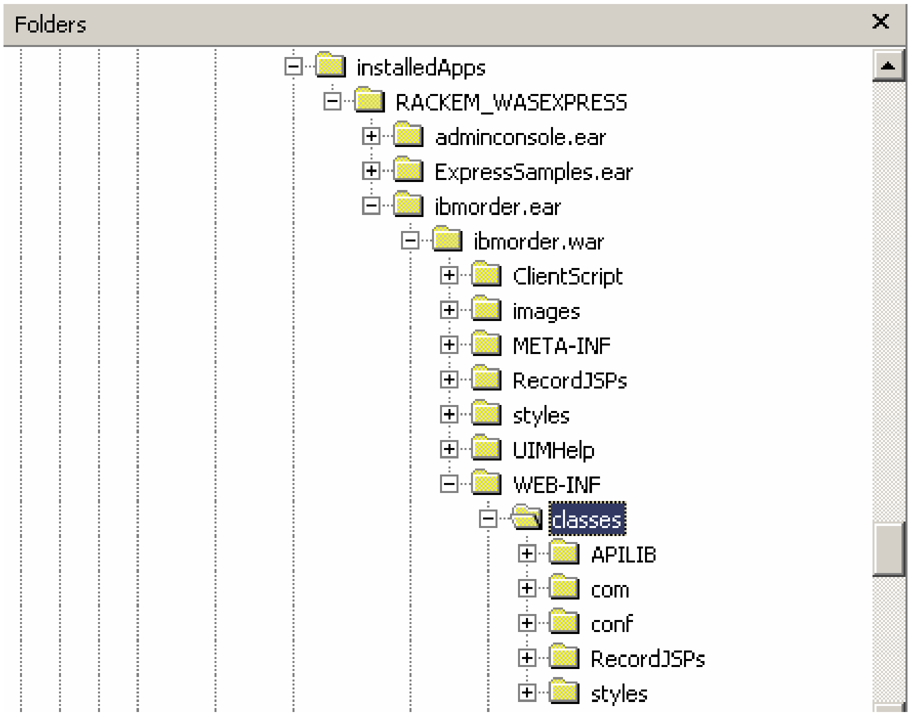


Figure 19 — Directory Structure of precompiled JSPs with precompiled JSPs

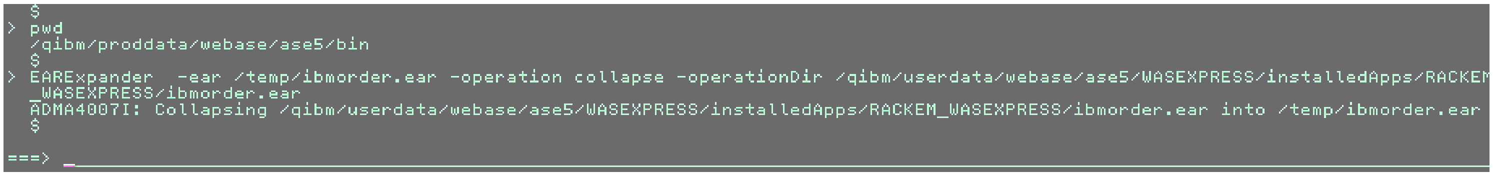
Notice how Figure 18 differs from Figure 19. The RecordJSPs and styles directories now exist under the \QIBM\UserData\WebASE\ASE\MYINSTANCE\installedApps\MYSYSTEM_MYINSTANCE\MYAPP.ear\MYAPP.war\WEB-INF\classes directory. At server startup, WebSphere Application Server V4 and V5 will check for up-to-date compiled JSPs from this directory structure and use them if they are up-to-date. If not, it will recompile the JSP and place them below the temp directory tree.

Use the EARExpander tool repackage .EAR file

The EARExpander tool shipped with WebSphere Application Server provides the capability to create a new .EAR file from a deployed application. In effect to collapse the deployed application, the tool is located in the /QIBM/ProdData/WebASE/ASE5/bin directory for WebSphere Application Server - Express V5 for iSeries. Creating a new .EAR or .WAR file from the deployed application is relatively easy but there are a couple of steps along the process.

1. Create a temporary directory to hold the collapsed ear file
2. Start QSHELL
3. Change directories to /QIBM/ProdData/WebASE/ASE5/bin
4. Execute the EARExpander tool:
   EARExpander –ear /temp/MYAPPLICATION.ear –operation collapse -operationDir/QIBM/UserData/WebASE/ASE5/installedApps/MYSYSTEM_MYINSTNACE/MYAPPLICATION.ear

Running this command will then compress all of the directories and files for the installed application and create a new .EAR file in the directory where the command was run. Figure 20 is a screen shot of QSHELL running this command.



```
  $
> pwd
  /qibm/proddata/webase/ase5/bin
  $
> EARExpander  -ear /temp/ibmorder.ear -operation collapse -operationDir /qibm/userdata/webase/ase5/WASEXPRESS/installedApps/RACKEM
  _WASEXPRESS/ibmorder.ear
  ADMA4007I: Collapsing /qibm/userdata/webase/ase5/WASEXPRESS/installedApps/RACKEM_WASEXPRESS/ibmorder.ear into /temp/ibmorder.ear
  $

===> _
```

Figure 20 — Example running EARExpander from QSHELL

In the example above, the result is an application in the /temp directory called ibmorder.ear which contains the deployed application from the specified directory. In this case since the precompiled JSPs were copied from the temp directory to the deployed application, the resulting .ear file will contain the precompiled JSPs.

Network Connection Speeds and Compression

LAN connection speeds and Internet hops can have a large impact on page response times. A fast server but slow LAN connection will yield slow end-user performance and an unhappy customer.

It is very common for a browser page to contain 15-60K of data. Customers who may have a current green-screen application over a 256K internet connection, running e-mail, Print, and other applications might find results of a WebFaced application unacceptable. If every screen averages 60K, the time for that data spend on the wire is significant. Multiply that by several users simultaneously using the application, and page response times will be reduced.

There are now two options available to support HTTP compression for Webfaced applications, which will significantly improve response times over a slow internet connection. As of July 1, 2003; compression support was added with the latest set of PTFs for IBM HTTP Server (powered by Apache) for iSeries (5722-DG1). Also, Version 5.0 of the IBM WebFacing Tool was updated to support compression available in WebSphere Application Server. On an iSeries server, the recommended WebSphere application configuration is to run Apache as the web server and WebSphere Application Server as the application server. Therefore, it is recommended that you configure HTTP compression support in Apache. However, in certain instances, HTTP compression configuration may be necessary using the IBM WebFacing Tool/WebSphere Application Server support.

The overall performance in both cases is essentially equivalent. Both provide significant improvement for end-user response times on slower Internet connections, but also require additional HTTP/WebSphere Application Server CPU resources. In measurements done with compression, the amount of CPU required by HTTP/WebSphere Application Server increased by approximately 25-30%. When compression is enabled, ensure that there is sufficient CPU to support it. Compression is particularly beneficial when end users are attached via a Wide Area Network (WAN) where the network connection speed is 256K or less. In these cases, the end user will realize significantly improved response times (see chart below). If the end users are attached via a 512K connection, evaluate whether the realized response time improvements offset the increased CPU requirements. Compression should not be used if end users are connected via a local intranet due to the increased CPU requirements and no measurable improvement in response time.

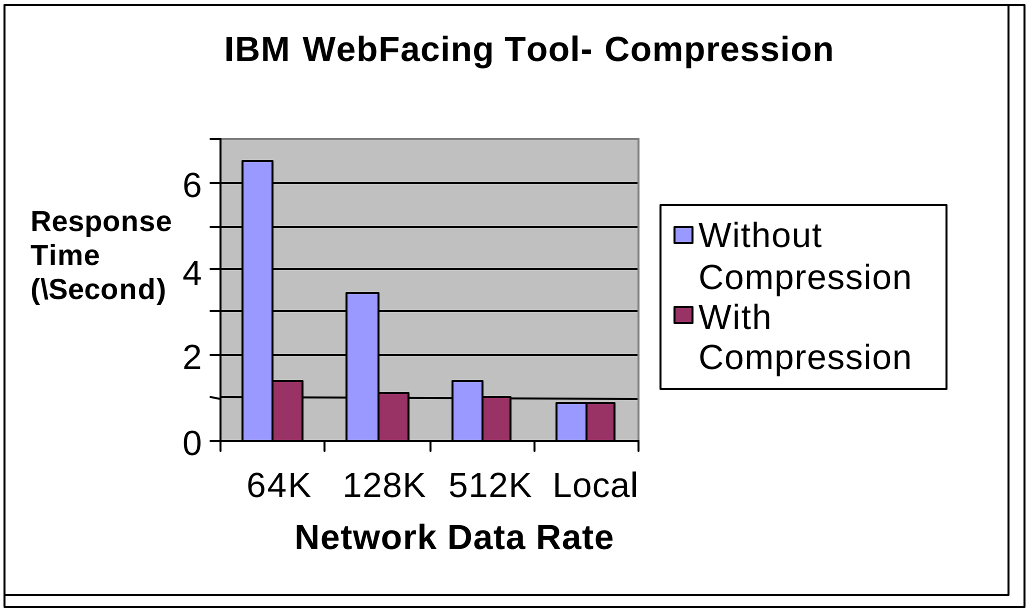


Figure 21 — IBM WebFacing Tool Compression

NOTE: The above results were achieved in a controlled environment and may not be repeatable in other environments. Improvements depend on many factors.

With the IBM WebFacing Tool V5.0, compression is ‘turned on’ by default. Customers in a local high speed intranet environment may want to change the configuration to ‘turn off’ compression in order to reduce the CPU utilization. This is particularly important if the CPU utilization of interactive types of users (Priority 20 jobs) is about 70-80% of the interactive capacity. In order to ‘turn off’ compression, edit the web.xml file for a deployed Web application. There is a filter definition and filter mapping definition that defines compression should be used by the WebFacing application (see below). These statements should be deleted in order to ‘turn off’ compression. In a future service pack of the WebFacing Tool, compression will be configurable from within WebSphere Development Studio Client.

```
<filter id="Filter_1051910189313">
        <filter-name>CompressionFilter</filter-name>
        <display-name>CompressionFilter</display-name>
        <description>WebFacing Compression Filter</description>
        <filter-
class>com.ibm.etools.iseries.webfacing.runtime.filters.CompressionFilter</filter-
class>
 </filter>
     <filter-mapping id="FilterMapping_1051910189315">
        <filter-name>CompressionFilter</filter-name>
        <url-pattern>/WFScreenBuilder</url-pattern>
 </filter-mapping>
```

Enabling Compression in IBM HTTP Server (powered by Apache)

The HTTP compression support was added with the latest set of PTFs for IBM HTTP Server for iSeries (5722-DG1). For V5R1, the PTFs are SI09287 and SI09223. For V5R2, the PTFs are SI09286 and SI09224.

There is a LoadModule directive that needs to be added to the HTTP config file in order to get compression based on this new support. It looks like this:

```
LoadModule  deflate_module
/QSYS.LIB/QHTTPSVR.LIB/QZSRCORE.SRVPGM
```

The following directive will also need to be added to the container to be compressed, or globally if the compression can always be done:

```
SetOutputFilter DEFLATE
```

There is mod_deflate documentation on the Apache Web site (http://httpd.apache.org/docs-2.0/mod/mod_deflate.html) that has information specific to setting up for compression. This is the best place to look for details. The LoadModule and SetOutputFilter directives are required for mod_deflate to work. Any other directives are used to further define how the compression is done.

Since the compression support in Apache for iSeries is a recent enhancement, the iSeries Information Center documentation for the HTTP compression support is not currently available (as of September 2003). See the IBM HTTP Server for iSeries Web site (http://**ibm.com**/eserver/iseries/software/http/) for when this documentation has been completed. Until the documentation is available, the information at: http://httpd.apache.org/docs-2.0/mod/mod_deflate.html can be used as a reference for tuning how mod_deflate compression is done.

Enabling Compression using IBM WebFacing Tool and WebSphere Application Server Support

You would configure compression using the IBM WebFacing Tool/WebSphere support in environments where the internal HTTP server in WebSphere Application Server is used. This may be the case in a test environment or in environments running WebSphere Application Server — Express V5.0 on an xSeries Server.

With the IBM WebFacing Tool V5.0, compression is 'turned on' by default. This should be 'turned off' if compression is configured in Apache or if the LAN environment is a local high speed connection. This is particularly important if the CPU utilization of interactive types of users (Priority 20 jobs) is about 70-80% of the interactive capacity. To 'turn off' compression, edit the web.xml file for a deployed Web application. There is a filter definition and filter mapping definition that defines compression should be used by the WebFacing application (see below). These statements should be deleted in order to 'turn off' compression. In a future service pack of the IBM WebFacing Tool, it is planned that compression will be configurable from within the WebSphere Development Studio Client.

```
<filter id="Filter_1051910189313">
        <filter-name>CompressionFilter</filter-name>
        <display-name>CompressionFilter</display-name>
        <description>WebFacing Compression Filter</description>
        <filter-
class>com.ibm.etools.iseries.webfacing.runtime.filters.CompressionFilter</filter-
class>
 </filter>
     <filter-mapping id="FilterMapping_1051910189315">
        <filter-name>CompressionFilter</filter-name>
        <url-pattern>/WFScreenBuilder</url-pattern>
 </filter-mapping>
```

PC Client

The PC client has a bigger impact than most think. Many people think that the PC client is just running a browser so an old Pentium® II processor should be enough. However, a slow client PC can degrade the end-user response times of an application significantly. Testing of the order entry application showed a 1/2 second page response time reduction on a slow client PC. This may mean that a 50% improvement in performance can be realized using the same application, same network, but a faster PC client. In the timings section listed below, there are some interesting results between a client Pentium II 450 MHz machine and a 1GHz Athelon. A client PC with a minimum of a Pentium III 600 MHz 512 MB RAM should eliminate most client performance issues.

The fastest data transmitted is the data that is never transmitted. This can be accomplished by making sure browser settings on the client are caching data properly. HTML pages are really a series of requests for text and images. Much of this data can be cached at the browser instead of being downloaded every time. When testing, make sure that browser caching is turned on. This is done by selecting the "Check for newer versions of stored pages: Automatically within Internet Explorer Temporary Internet files" configuration button.

Timings

This section has some performance numbers to demonstrate much of what was discussed earlier. These timings were taken in an unsophisticated way. A stop watch was used between screen transitions and the number of seconds recorded. While not sophisticated, the data is accurate enough to get a good idea of the improvements that can be made on an application using the techniques described above.

Timing #1 — Demonstrate time required to compile JSP files and build Byte Code Cache file of an IBM WebFacing Web-enabled Application
System — Model 800 FC# 0863 300 CPW, OS Version — V5R2
WebSphere Application Server - Express V5, 41MB Application, 96M Heap 1G RAM, 890M In Base Memory Pool
IBM WebFacing Tool V4
100MB LAN Connection

All Tests taken after initial server startup. All tests have JIT disabled.	No Precompiled JSPs No ByteCode Cache File Data A	Precompiled JSPs No ByteCode Cache File Data B	Precompiled JSPs. Creating ByteCode Cache File Data C	Precompiled JSPs Use ByteCode Cache File Data D
Page After Logon	Minutes	13	125	17
Security Screen	Minutes	3	21	1
Pick A Company	Minutes	10	21	5
Change Orders	Minutes	3	11	2

The timings above are to demonstrate the time required to both compile the JSPs, and create the ByteCode Cache File on a first touch of a JSP. These timings demonstrate the need to make sure JSPs are precompiled to achieve acceptable first touch user response times Notice that using the ByteCode Cache file does have a positive impact on first touch of JSP files except in the first case which should be disregarded because of additional processing occurring by the application.

Timing #2 — Demonstrate second and subsequent JSP touches of an IBM WebFacing Web-Enabled Application

System — Model 800 FC# 0863 300 CPW, OS Version — V5R2
WebSphere Application Server - Express V5, 41MB Application, 96M Heap 1G RAM, 890M In Base
IBM WebFacing Tool V4
100MB LAN Connection

All tests taken JSP touched one-time (Meaning JSP was class loaded and ByteCode verified) All tests have JIT disabled.	Precompiled JSPs Data A	Precompiled JSPs Using ByteCode Cache File Data B	Second Touch Data C
Page After Logon	13	17	9
Security Screen	3	1	1
Pick A Company	10	5	2
Change Orders	3	2	3

The timings above are to demonstrate that the second and all subsequent touches of a JSP should perform faster than the first touch of a JSP. This is due to the time it takes for the JSP class to be loaded into the WebSphere Application Server and perform the ByteCode verification when the ByteCode Cache file is not used.

Timing #3 - Demonstrate memory variable effects of an IBM WebFacing Web-enabled Application for a Single User System — Model 800 FC# 0863
300 CPW, OS Version — V5R2
WebSphere Application Server — Express V5, 41MB Application, 96M Heap
IBM WebFacing Tool V4
100MB LAN Connection

All tests did not use ByteCode Cache File. All tests used Precompiled JSP Files All tests have JIT disabled. All times are in seconds	512MB Total Memory. 390MB In WebSphere Application Server Memory Pool. First Touch Data A	512MB Total Memory. 390MB In WebSphere Application Server Memory Pool. Second Touch Data B	1G Total Memory. 890MB In WebSphere Application Server Memory Pool. First Touch Data C	1G Total Memory. 890MB In WebSphere Application Server Memory Pool. Second Touch Data D
Application Startup	300		180	
Page After Logon	77	11	13	13
Security Screen	3	1	3	3
Pick A Company	12	2	10	4
Change Orders	4	2	3	2

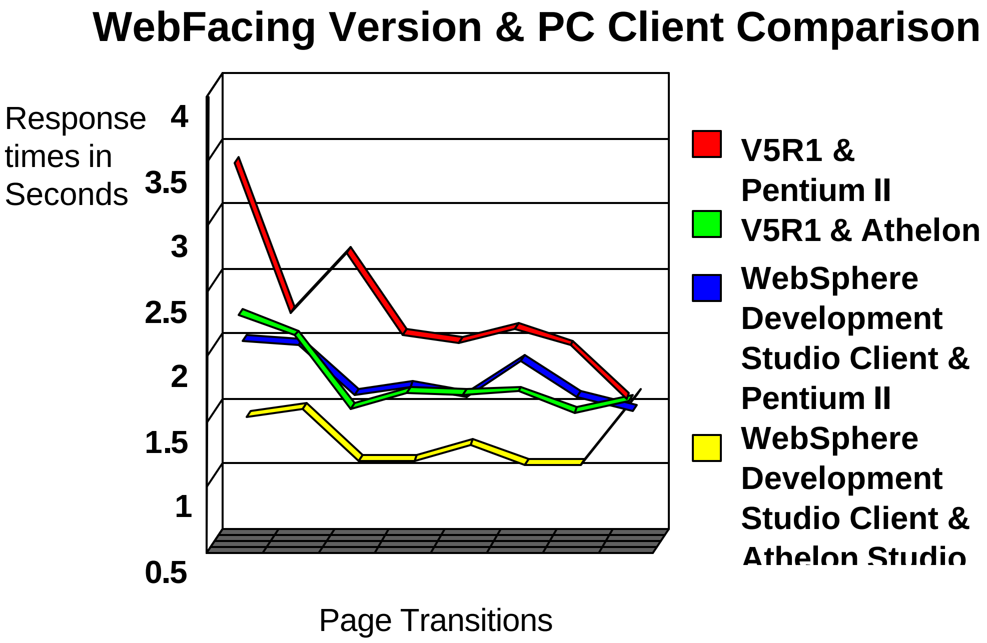
The timings above are to demonstrate that the added memory to the pool where WebSphere Application Server is running had a significant impact on both the application startup and user response times, both on the first and subsequent touches. Response times on second touch for Data B may be acceptable for a single user. Data in Timings #4 demonstrates with added users that 512MB of memory will not be acceptable for most users.

Timing #4— Demonstrate timing differences between V5R1 IBM WebFacing Web-enabled Application and WebSphere Development Studio Client on two different Client PC types.

System — Model 820 FC# 24AA 1100 CPW
OS Version — V5R1
WebSphere Application Server V4.0 1.5GB Base Memory Pool 256MB Heap
10MB LAN Connection

KEY:
V5R1 - V5R1 Version of WebFacing Tool
Pentium II - Pentium II 450MHZ 128M RAM
Athelon - 1GHZ 500MB RAM

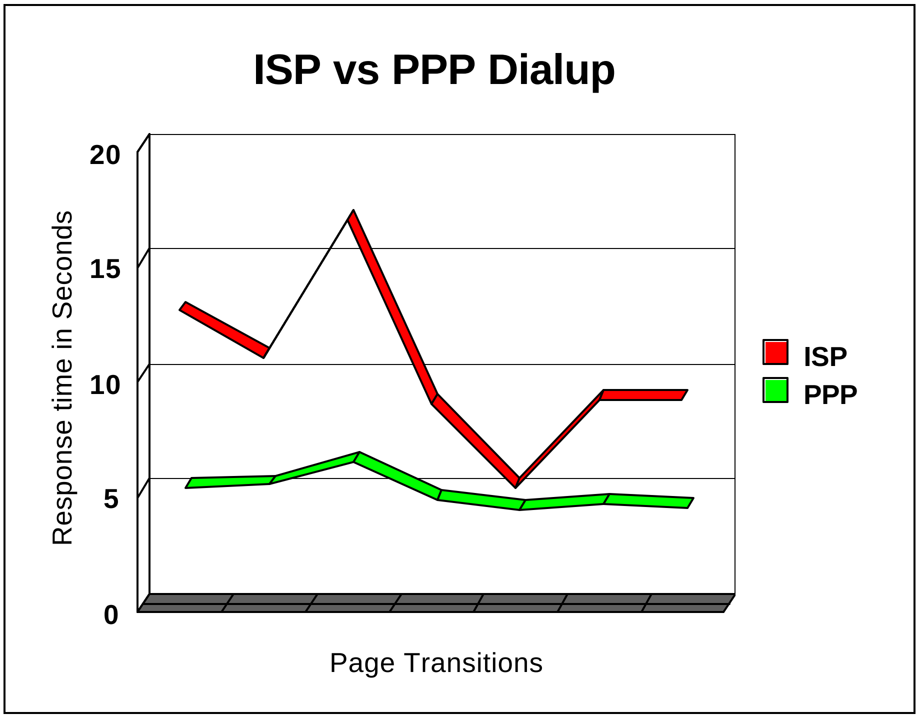
	V5R1 Pentium II	V5R1 Athelon	WebSphere Development Studio Client V4 Pentium II	WebSphere Development Studio Client V4Athelon
Customer Prompt	3.5	2.28	2.03	1.4
Select Customer	2.34	2.12	2	1.47
Part Number Prompt	2.81	1.56	1.62	1.06
Select Part Number	2.18	1.68	1.69	1.06
Change Part Quantity	2.11	1.66	1.6	1.19
Accept Order	2.22	1.69	1.88	1.03
Invalid Customer	2.09	1.53	1.6	1.03
Exit	1.69	1.62	1.5	1.57



In this timing, both of the versions of the Web-enabled application and the PC client had a significant impact on performance. In general, there was 1/2 a second improvement using the WebSphere Development Studio Client version of the WebFacing Tool, and an additional 1/2 second improvement using a PC client of near modern capabilities. Clearly, using a combination of WebSphere Development Studio Client and a well performing PC client provides better performance.

Timing #6 — Demonstrate timing differences Point-to-Point and Dial-up ISP Connection
System — Model 270
OS Version — V5R1
WebSphere Application Server V3.5.6
WebSphere Development Studio Client early release

	28K Dial-up using ISP	24K Dial-up Point-to-Point Connection
Customer Prompt	13.14	5.01
Select Customer	11.1	5.15
Part Number Prompt	17.06	6.13
Select Part Number	9.12	4.5
Change Part Quantity	5.44	4.03
Accept Order	9.27	4.29
Invalid Customer	9.3	4.17



The data collected in this example clearly shows the negative impact a poor ISP connection can have on performance. The Point-to-Point connection performed consistently better than the connection through an ISP. This does not mean that all ISP providers will provide poor performance. It demonstrates that performance can be severely limited when the Internet connection is inadequate.

Conclusion

As this paper has shown, Web-based applications are different than traditional green-screen applications. They require more horsepower at both the server and client and transmit more data on the LAN. They utilize a connectionless protocol, so there is no typing ahead like there is in green-screen applications. Because of this, users who expect green-screen performance from Web-based applications may be disappointed.

Can a WebFaced application perform as well as other browser-based applications? Absolutely. There is no reason why one- to three-second screen transition times should not be normal on properly configured server, LAN, client hardware, and browser cache. If one- to three-second page transition response times are acceptable to a customer, the benefit of a browser-based application and freshness of a GUI would be a great addition to any company’s software portfolio.

Special Thanks

A BIG thank you to the people who have provided input and reviews for this paper. The combined knowledge and expertise of these people provided a much more detailed and broader scope to the topics covered than could have been done by any one individual.

Byron Bailey	- IBM Rochester
Jim Beck	- IBM Rochester
Charles Farrell	- IBM Rochester
Scott Moore	- IBM Rochester
Michael Sandberg	- IBM Rochester

Reference Web Sites

IBM eServer Enablement
ibm.com/servers/enable

IBM Information Center
http://publib.boulder.ibm.com/pubs/html/as400/infocenter.html

WebSphere Development Studio Client for iSeries
ibm.com/software/ad/wds400/

WebSphere Development Studio Client for iSeries Support page
ibm.com/software/ad/wds400/support/

WebSphere Application Server for iSeries
ibm.com/eserver/iseries/software/websphere/wsappserver/

WebSphere Application Server - Express V5 for iSeries
http://publib.boulder.ibm.com/iseries/v5r2/ic2924/index.htm?info/rzamy/50/express.htm

WebSphere Application Server V4.0 Advanced
http://publib.boulder.ibm.com/was400/40/AE/english/docs/pvindex3.html

WebSphere Performance Considerations
ibm.com/eserver/iseries/software/websphere/wsappserver/product/performancews50.html

Trademarks

IBM, eServer, iSeries, xSeries, pSeries, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States and other countries.

Intel and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Microsoft and Windows are registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

All other registered trademarks and trademarks are properties of their respective owners.

IBM makes no commitment to make available any products referred to herein.

References in this publication to IBM products or services do not imply that IBM intends to make them available in every country in which IBM operates.