IBM® WebSphere® Developer Technical Journal

to go!

Issue 15.5
August 1, 2012

Inside:

- Using IBM Worklight to develop cross-platform HTML5 video play hybrid applications
- System administration in WebSphere Application Server V8.5 Part 3: High Performance Extensible Logging (HPEL)
- The importance of embracing the performance lifecycle
- Performance testing and analysis with WebSphere Application Server
- Key features and capabilities of IBM Worklight to accelerate your mobile development

Plus:

- *The Support Authority*

IBM

IBM WebSphere Developer Technical Journal

Issue 15.5 : August 1, 2012

developerWorks.

From the editor

In this issue of the **IBM® WebSphere® Developer Technical Journal** our coverage of IBM Worklight continues with a look at how you can build successful cross-platform hybrid apps that play video without letting HTML5's shortcomings get in the way. Our series on system management in IBM WebSphere Application Server V8.5 continues with a look at the enhanced capabilities of the HPEL logging and tracing mechanism. Performance testing is also a featured topic, first with information to help get stakeholders behind the performance lifecycle, and then with information to help you implement it with WebSphere Application Server. Plus, The Support Authority introduces the new cloud-based IBM Support Assistant, and more.

Your required reading begins below…

Feature articles

- **Using IBM Worklight to develop cross-platform HTML5 video play hybrid applications**
  by Bill Paris, Sreeni Pamidala and Raghunandan K Harithas
  With the rush to extend enterprise access beyond computers to mobile devices, mobile hybrid apps are being used to take advantage of HTML5's cross-platform capabilities. Unfortunately, support for HTML5 falls short when it comes to cross-platform video play, particularly in hybrid apps running on the Android operating system. This article shows how you can work around those issues and enable video play by leveraging the mobile hybrid capabilities of IBM Worklight.

- **System administration in WebSphere Application Server V8.5 Part 3: High Performance Extensible Logging (HPEL)**
  by Don Bourne, Igor Belyi and Scott Highbarger
  IBM WebSphere Application Server V8.0 introduced a new way to store and work with log and trace content called High Performance Extensible Logging (HPEL). As the name suggests, HPEL was created to significantly outperform the existing application server logging and tracing mechanism, plus provide a powerful way to extend log and trace entries with extra fields. HPEL was further improved in the newly released WebSphere Application Server V8.5, providing a number of ease of use improvements for working with log and trace content, making it even more useful. Enabling HPEL is quick and easy, and does not require any code changes to your applications. This article is a primer for administrators considering trying HPEL, and explains what HPEL is all about and why it matters.

- **The importance of embracing the performance lifecycle**
  by Keri-Anne Lounsbury and Kevin Yu
  Pretty much everyone will agree that performance testing is important, but scenarios where performance testing is implemented in an effective way are few and far between. In some cases this is because the commitment of time and resources — and the commitment from stakeholders — is difficult to obtain, and in other cases there is a lack of understand of what "performance" actually means. This overview of the fundamental concepts behind the performance lifecycle will help you understand the

activities and involvements needed from various stakeholders to ensure a successful implementation.

- **Performance testing and analysis with WebSphere Application Server**
  by David Hare
  And speaking of performance testing, here is an approach that puts the performance lifecycle into action. IBM WebSphere Application Server supports an ever-growing range of applications, each with their own unique set of features, requirements, and services. Proper performance testing and analysis on each of these applications is essential to ensure they are performing at their maximum potential. This article provides guidance on some best practices on how to build a performance test, compare results across application or environment changes, and how to identify bottlenecks using freely available tools from IBM. The methodologies described here apply to all versions of WebSphere Application Server, including the newly released WebSphere Application Server V8.5.

- **Key features and capabilities of IBM Worklight to accelerate your mobile development**
  by Harish Shenoy
  IBM Worklight provides a complete platform for developing, deploying, hosting, and managing mobile enterprise applications. Worklight addresses all the requirements for mobile application developments and provides tools and efficiencies to help at every stage of the development and deployment process. This article offers an overview of IBM Worklight V5 with a checklist of some of the new and key features that will help you launch and achieve your mobile development objectives.

The Support Authority

- **An overview of the IBM Support Assistant 5.0 Beta**
  by Jim McVea and Paul Blizniak
  IBM Support Assistant is a free application that provides features for self-help problem determination and a platform for obtaining diagnostic tools. These features and tools streamline troubleshooting by making it easy to organize, analyze, diagnose, and eventually solve problems that might occur with IBM software or deployed applications. The new Beta release of IBM Support Assistant 5.0 features many usability improvements over the current version, plus it's been developed with a cloud-based approach. This article introduces the IBM Support Assistant 5.0 strategy and offers a basic understanding of the tool, its key concepts, and core functionshosted by an XC10 caching appliance without requiring any application code changes.

Using IBM Worklight to develop cross-platform HTML5 video play hybrid applications

Bill Paris (williamparis@us.ibm.com)
Software Developer
IBM

Sreeni Pamidala (spamidala@us.ibm.com)
Senior Certified IT Architect
IBM

Raghunandan K Harithas (rnandan@us.ibm.com)
Consulting IT Specialist
IBM

Skill Level: Intermediate

Date: 01 Aug 2012

With the rush to extend enterprise access beyond computers to mobile devices, mobile hybrid apps are being used to take advantage of HTML5's cross-platform capabilities. Unfortunately, support for HTML5 falls short when it comes to cross-platform video play, particularly in hybrid apps running on the Android operating system. This article shows how you can work around those issues and enable video play by leveraging the mobile hybrid capabilities of IBM® Worklight.

Introduction

A mobile hybrid app combines native operating system functionality with web technologies. Typically, a hybrid app presents content in an embedded web browser, an approach that enhances cross-platform capabilities because the majority of the code can be written using HTML5 technologies, while at the same time enabling access to native device features when necessary. **IBM Worklight** is a mobile application platform that enables the development of cross-platform hybrid applications, offering mechanisms to navigate between the web view and the native view, and providing a development and run time environment that moves hybrid apps much closer to the write-once-run-anywhere goal.

Video play within an app is one area where a cross-platform capability can be difficult to achieve. HTML5's <video> tag was intended to enable cross-platform video play, but inconsistent support for its capabilities fall short of that goal.

This article focuses on developing video play within cross-platform HTML5 hybrid apps and explains how the IBM Worklight development platform can help work around problems when support for HTML5's video capabilities falls short on any one particular platform.

The challenges of video play

Video play on computers and mobile devices has long posed challenges to developers because development with video requires an understanding of complex technology and terminology. Before going too much further, let's get some terminology out of the way.

Video is a combination of one video (or picture) stream and one or more audio streams, all packed into a single archive. In common video parlance:

- A **video container** wraps audio and video files together into a single file archive. There are lots of video container formats and some popular ones are MPEG4, Flash Video, Ogg, WebM, and Audio Video Interleave. The container format is indicated by the file extension, such as mp4, flv, ogv, webm, and avi along with others.
- A **video codec** identifies the software algorithm by which a video stream is encoded and decoded (compressed and decompressed). A video player needs to know which codec was used in order to decode and play the video stream.
- An **audio codec** is similar to a video codec, but for audio streams.

The move toward HTML5

A primary challenge of video play comes from the evolution of video play mechanisms in computers and mobile devices. Prior to the HTML5 specification, there was no standard way to play video in browsers, and almost always video was funneled through third-party plug-ins such as RealPlayer, Apple QuickTime, and Adobe Flash. HTML5's video playing capability came about in part to address the reliance on plug-ins even as the popularity of YouTube for video play made Flash the de-facto standard on desktops. With Apple's rejection of Flash in favor of HTML5 for video play on iOS devices, cross-domain video play using Flash became impossible and developers began to focus on creating websites that use HTML5 for video play.

HTML5 added a new video tag for directly embedding video content in a web page to play video without the use of plug-ins. Listing 1 show an example of a video tag to play an mp4 video in a browser window. It specifies the width and height, the amount of space allocated to the video player embedded within the browser window, that autoplay is enabled so the video will start playing without user action, and that onscreen video controls (play, pause, volume) are to be enabled. Most importantly, it specifies the video source archive to be played.

Listing 1. HTML5 video tag for embedding video in a webpage

```
<video width="320px" height="480px" autoplay="autoplay" controls="controls">
 <source src="dir/video.mp4" type="video/mp4"/>
</video>
```

The adoption of the HTML5 video tag has been encouraging. Initially, Safari was the only browser offering HTML5 video support when the HTML5 specs were drafted, but now all modern browsers support it. Streaming video on the Web is in the midst of a fast transition from Flash plug-ins to the HTML5 standard, even though the HTML5 specification is still in draft form, with expected completion in 2014.

Where HTML5 video play falls short

Unfortunately, browser support for the HTML5 video tag is only part of the story because video play requires more than just tag support. Support for video container formats, video and audio codecs, and streaming protocols all play a big role in web page interoperability across browsers. For instance, browser support for codecs such as H.264 and WebM has been selective, with some browsers supporting one but not the other. HTML5 doesn't specify a codec because standards groups haven’t agreed on a single one, which means web developers contemplating the use of HTML5 video must consider browser compatibility issues.

Additional challenges come about with mobile video play, due to the diverse nature of mobile devices with their differing screen resolutions and processors. For example, there are hundreds of Android variants in the mobile market, all of which need to play audio and video.

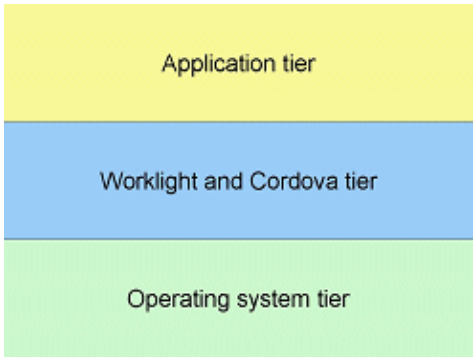
Trials, tribulations, and a solution

Our SmarterTVApp project involved the development of a cross-platform mobile hybrid app with video playing capability, and it seemed only natural to use HTML5 video for video play. We put together a development environment using Eclipse augmented with the Worklight Studio 4.2 Eclipse plug-in, server hardware running Worklight Server on Red Hat Linux®, and a video storage server populated with MPEG4 video files. Multiple mobile devices running Android 4.0 Ice Cream Sandwich and Apple iOS 5 were used for testing, and they received video from a video storage server which streamed video using the HTTP Level Streaming (HLS) protocol.

Three-tiered hybrid app model

Worklight-developed apps use a three-tiered cross-platform application model, illustrated in Figure 1. The lowest tier consists of the native operating system functionality provided by native operating system API libraries. For our SmarterTVApp, this consisted of Android or iOS operating system API invocation points that were built into the app.

Figure 1. Three-tiered application model



The middle tier consists of the Worklight and Apache Cordova components supplied by Worklight which bridge the HTML5 application code with native device operating system functionality. Apache Cordova (formerly known as PhoneGap) is an open-source mobile development framework that enables the development of hybrid apps by providing access to native operating system capabilities through JavaScript™. The Worklight components in this tier provide client-side functionality, such as device skinning, encrypted storage, push notifications, server integration framework, and many other capabilities.

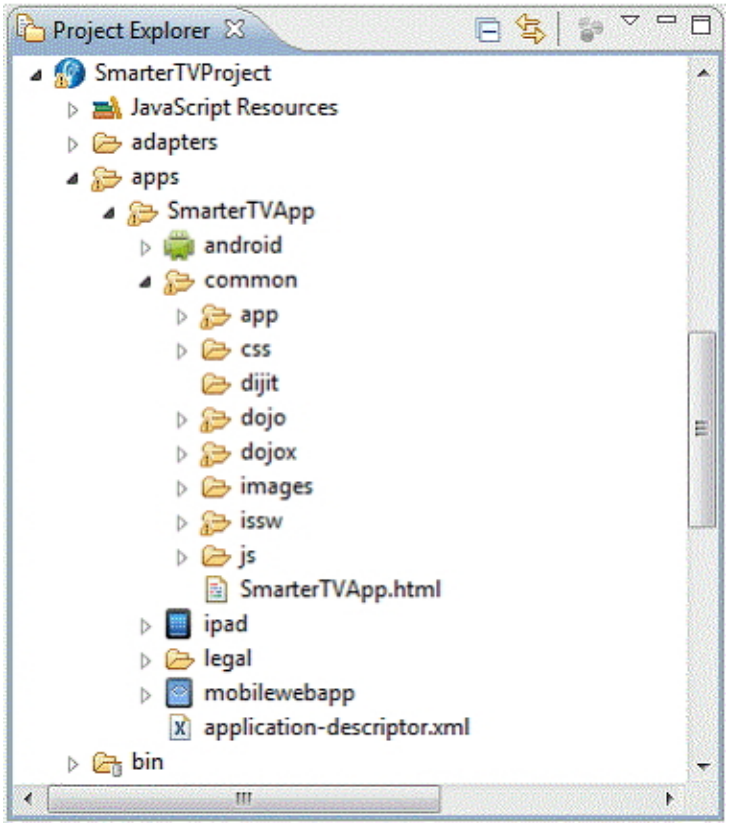
The top tier consists of application components. In the SmarterTVApp, this consisted of our custom application JavaScript, HTML, and CSS code, along with components we imported to support our application, the IBM Dojo Toolkit release 1.7.2, and the IBM issw.mobile application framework used by our app for presentation and view-to-view transitioning.

Development environment

When a Worklight project is created in Eclipse, the Worklight Studio plug-in creates an initial directory structure populated with a set of folders for the application and runtime environment. The directory structure for SmarterTVApp is shown in Figure 2. The common folder holds the JavaScript, HTML, and CSS files common to all device deployment environments while the android, ipad, and mobilewebapp folders hold device specific optimization files.

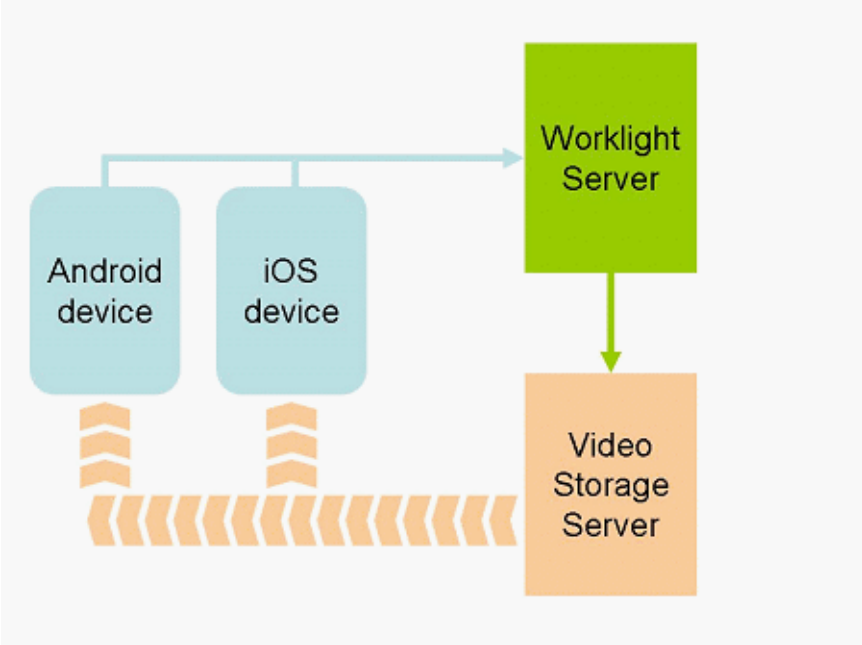
Under the common folder, we also added Dojo related folders (dijit, dojo and dojox) to house the Dojo files used by our application, and added an issw folder for the files comprising the issw.mobile application framework.

Figure 2. Worklight project folders



A simplified illustration of the development test environment is shown in Figure 3. The Android and iOS devices interact with a server application running on the Worklight Server, which then initiates the streaming of video from a Video Storage Server to the devices.

Figure 3. The development test environment



Initial implementation using HTML5 video

Our initial design made use of HTML5 video to achieve cross-platform video play, so we implemented the video element with two source elements, as shown in Listing 2. When encountering multiple source elements, browsers will use the first recognized format, and our expectation was that the MP4 and OGV formats would provide sufficient cross-browser capability.

Listing 2. HTML5 video tag specifying a video in multiple formats

```
<video width="320px" height="480px" autoplay="autoplay" controls="controls">
 <source src="dir/video.mp4" type="video/mp4/">
 <source src="dir/video.webm" type="video/webm/">
</video>
```

Our high hopes for quickly implementing cross-platform video play through the use of HTML5 video were dashed as soon as we began testing our first prototype. With a few tweaks we succeeded in getting video playing on our app running on iOS devices, but we had no such luck when running it on Android. Thus began the trial and error phase of our development.

Working toward a solution

An important consideration here is that the mobile hybrid approach for Android, as implemented by Cordova, is based on the Android WebView class for displaying web pages. A Cordova-based hybrid app uses a WebView to display the HTML5 content portion of the app. In our testing, we found that while our HTML5 video content would play successfully in the Android browser, it would not play when the HTML5 video content was displayed through the hybrid app's WebView object.

We naturally sought information on the Web and one quick search confirmed our fears: the tech forums were full of postings from anguished developers facing similar WebView HTML5 video play problems on Android 4.0. Armed with the suggestions we read about, we tried a variety of video formats, tried changes to the video tag parameters, and tried changing many dynamic (JavaScript) and static (HTML) aspects of creating the video element and its attributes -- all without success.

So it was time for Plan B: abandon cross-platform video play using the HTML5 video tag and instead use Android native video play functionality when the app was built to run on Android. When built to run on iOS, the app would use HTML5 video play as originally planned.

One benefit of developing with Worklight is that it provides a simple mechanism to incorporate native pages into hybrid apps. Worklight's hybrid coding functionality enables an app to navigate between web and native pages, and to share data between these pages. By making use of this feature, our hybrid app had the ability to switch to a native page that made use of Android's Java API set to play video, and then when video play was completed, would switch back to the JavaScript code that was common to both iOS and Android.

With this knowledge, we set about implementing a native page that created an Android Activity using a VideoView object to play video, and at last we had success: video play when running on Android.

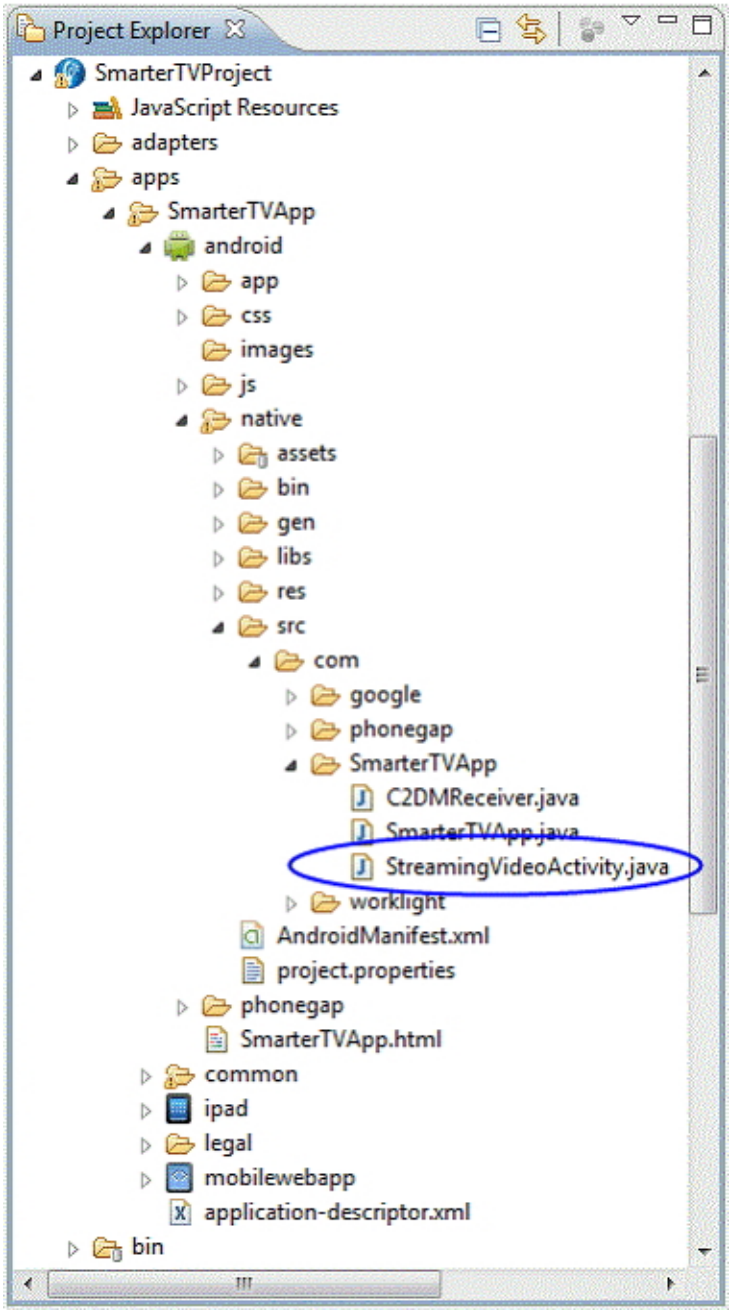
Solution in detail

The solution consists of both Android native Java code and JavaScript code. The JavaScript code uses Worklight's capabilities to invoke the Android Java code to play the video.

Android native Java code to play video

To implement the Java video functionality, we first created a new StreamingVideoActivity.java class under the Android specific project structure location, shown in Figure 4. The Worklight project structure (separating device specific optimization files into separate folders) simplified the addition of the Android native code required by our solution.

Figure 4. Worklight project folder containing Android native Java code



The StreamingVideoActivity is implemented as an extension of an Android Activity and plays video using the VideoView and MediaController classes. An outline of this implementation is shown in Listing 3. A complete solution would implement callbacks for handling video termination and completion events so as to return control back to Worklight's web-based screen view.

Listing 3. StreamingVideoActivity.java implementation

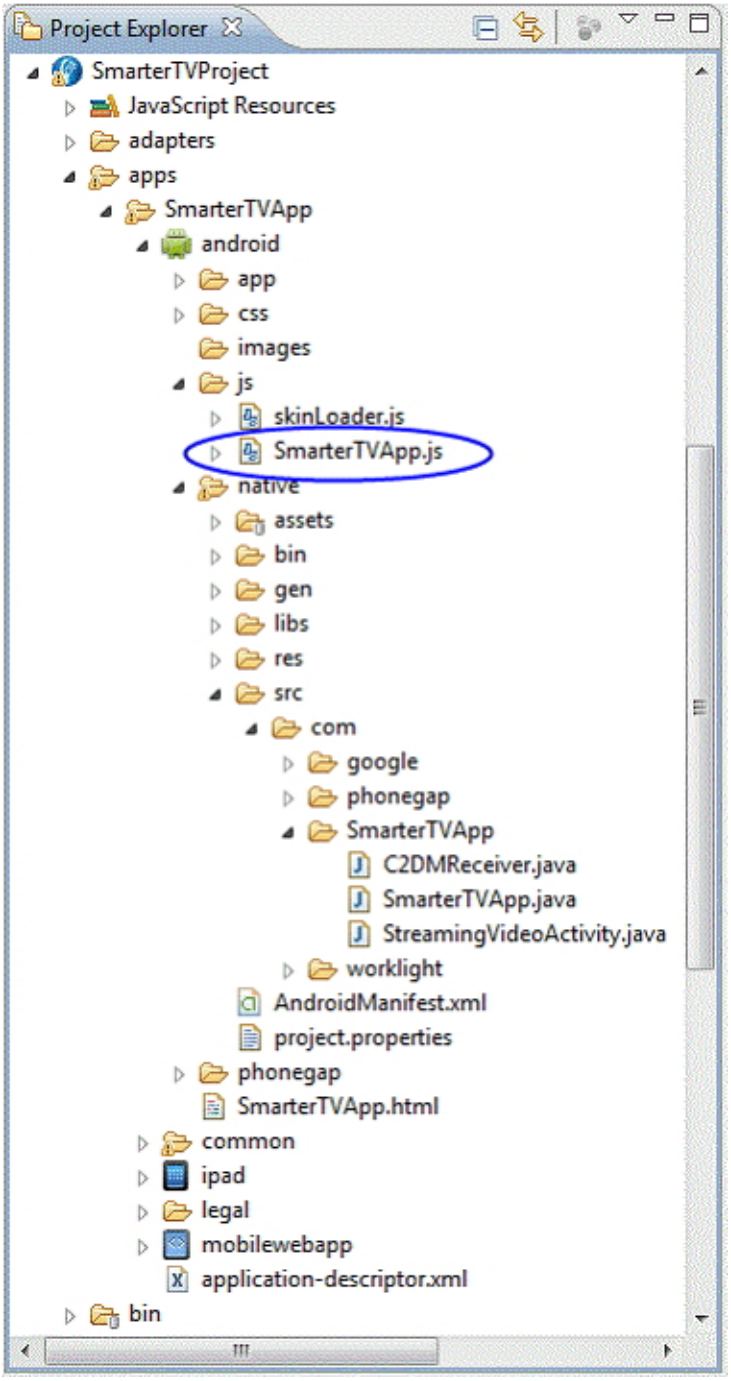
```
public class StreamingVideoActivity extends Activity {

    /** Called when the activity is first created. */
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {

 super.onCreate(savedInstanceState);
 Log.d("StreamingVideoActivity", "Entering onCreate");

 // Extract the URL from the Intent
 String url = getIntent().getStringExtra("urlParam");

 Log.d("StreamingVideoActivity", "About to play URL: url");
 VideoView videoView = new VideoView(this);
 videoView.setVideoPath(url);

 MediaController ctlr = new MediaController(this);
 ctlr.setMediaPlayer(videoView);
 videoView.setMediaController(ctlr);

 setContentView(videoView);
 videoView.requestFocus();
 Log.d("StreamingVideoActivity", "Leaving onCreate");
    }
}
```

Invoking the Android code from the web application

With the Android work complete, all that remained was to create a JavaScript function that uses the Worklight WL.NativePage.show API function to switch from the web-based Activity to the new native Android Activity. The JavaScript file SmarterTVApp.js implementing this functionality was placed in the Android specific project structure location shown in Figure 5.

Figure 5. Worklight project folder containing Android specific JavaScript file



The code excerpt from SmarterTVApp.js in Listing 4 shows the implementation of the openNativePage function that invokes the Worklight WL.NativePage.show function to run the StreamingVideoActivity Activity.

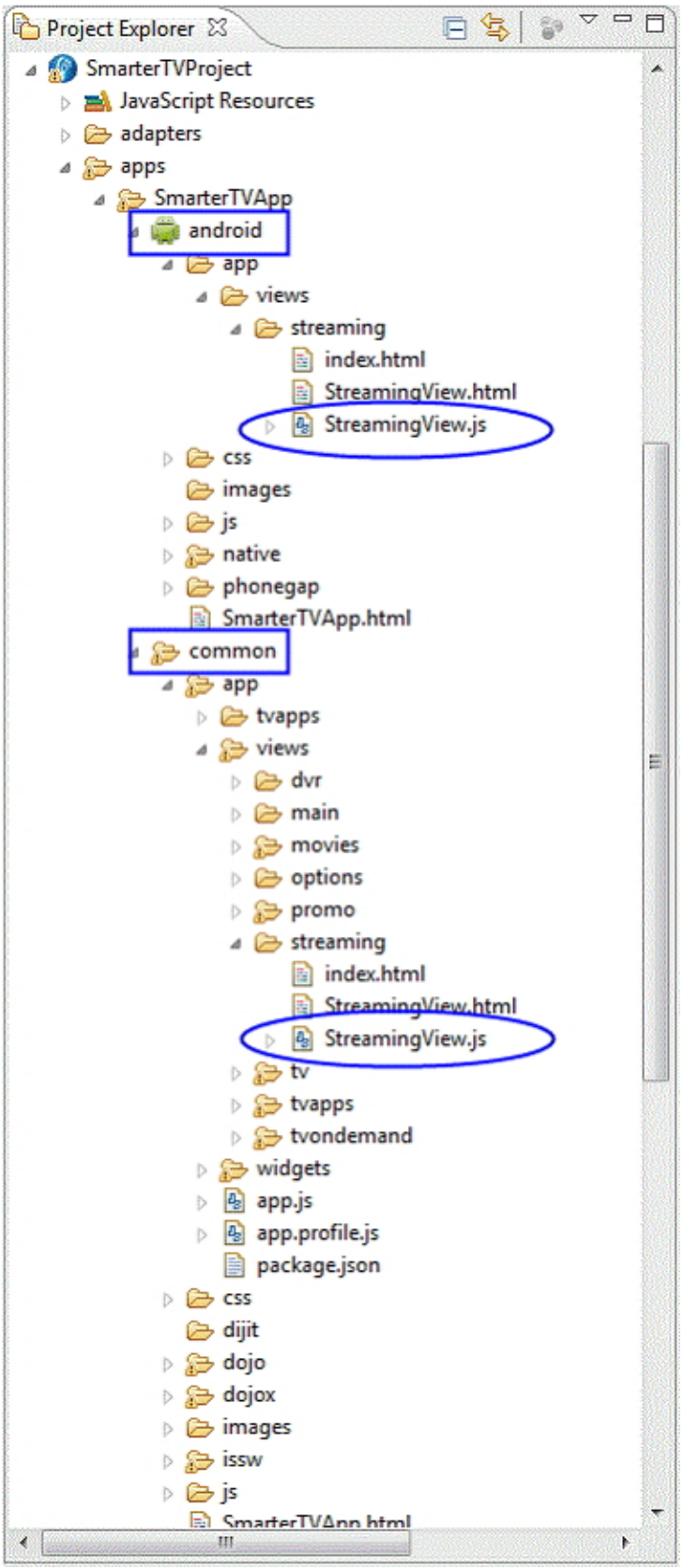
Listing 4. SmarterTVApp.js code excerpt

```
/**
 * Plays the specified video in an Android native page
 * @param url  The video URL
 */
```

```
function openNativePage (url) {
 WL.Logger.debug("Switching to SmarterTVApp.StreamingVideoActivity to play " +url);

 // Create an object to hold the URL.  The field name, urlParam, must match
 // the name used in the native Android Java code for extracting the URL
 var params = {urlParam : url};

 // Show the Android native page
 WL.NativePage.show('com.SmarterTVApp.StreamingVideoActivity',
  backFromNativePage, params);
}

/**
 * Invoked as a call-back on return from the Android native page
 * @param data
 */
function backFromNativePage(data) {
 WL.Logger.debug("Back from StreamingVideoActivity");
}
```

Finally, we needed to conditionally call openNativePage when the hybrid was running on an Android device, and this is where the Worklight Studio development environment was again helpful. Rather than detecting the device type and adding if-then-else JavaScript logic, the Worklight project structure handles this automatically. In our application video, play is carried out in the StreamingView.js file. By creating two versions of the file as shown in Figure 6, one stored in the common folder that initiates HTML5 video play and a second stored in the android folder that invokes the Worklight openNativePage function to initiate Android native video play, the Worklight Studio automatically handled the inclusion of the file version appropriate to the run time environment.

Figure 6. Dual implementations of video streaming



Conclusion

From this project we learned cross-platform video play can be difficult to achieve and the solution can require the use of native device functionality. Worklight simplifies development through the use of its capabilities for combining native and web-based functionality.

Acknowledgements

The authors wish to thank our colleagues Anton Aleksandrov, Raanan Avidor, Karl Bishop and Tom Thacher for their technical contributions and guidance that led to the success of this project.

Resources

Learn

- W3C HTML5 draft specification
- Apache Cordova (PhoneGap)
- Dive into HTML5, Mark Pilgrim
- What is H.264?
- The WebM Project
- IBM developerWorks Mobile zone
- IBM developerWorks WebSphere

Get products and technologies

- Download IBM Worklight Developer Edition
- Evaluate IBM Worklight Developer Edition 5.0

Discuss

- IBM Worklight Forums

About the authors

Bill Paris

Bill Paris is a software developer consulting for the IBM Software Services for WebSphere group. He specializes in the development of mobile applications and server middleware focused on the telecommunications industry.

Sreeni Pamidala

Sreeni Pamidala is a Senior Certified, IT Architect and currently working as a Lead Architect for ISSW Mobile Services, AIM Software group. In this role, he spends most of his time with customers across all industries to develop and deliver mobile solutions using IBM Mobile Foundation technologies that integrate with their enterprise infrastructure and middleware. Prior to this role, Sreeni worked with ComSector clients for 15 years where he led several successful engagements by developing and deploying complex web, network based carrier grade solutions for the mobile and legacy telephony service providers using IBM's cross-brand stack of software and hardware products and several third party vendors' integration products for the best of breed solutions.

Raghunandan K Harithas

Raghunandan Harithas is an IT specialist based out of the IBM Annapolis office in Maryland, USA. He has been working as a technical lead in several telecom projects using stacked WebSphere products. He is currently working in the field of mobile application development using the IBM Mobile Foundation software based on IBM Worklight.

System administration in WebSphere Application Server V8.5, Part 3: High Performance Extensible Logging (HPEL)

Don Bourne (dbourne@ca.ibm.com)
WebSphere Serviceability Architect
IBM

Igor Belyi (belyi@us.ibm.com)
Staff Software Engineer
IBM

Scott Highbarger (shighbar@us.ibm.com)
Advisory Software Engineer
IBM

Skill Level: Intermediate

Date: 01 Aug 2012

IBM® WebSphere® Application Server V8.0 introduced a new way to store and work with log and trace content called High Performance Extensible Logging. HPEL was further improved in the newly released WebSphere Application Server V8.5 making it even more useful. This article tells you what HPEL is all about, and why it matters.

View more content in this series

Introduction

IBM WebSphere Application Server V8.5 enhances the High Performance Extensible Logging (HPEL) feature that was first introduced in V8.0. As the name suggests, HPEL was created to significantly outperform the existing application server logging and tracing mechanism, plus provide a powerful way to extend log and trace entries with extra fields. Beyond that, HPEL provides a number of ease of use improvements for working with log and trace content. Enabling HPEL is quick and easy, and does not require any code changes to your applications.

This article provides a primer for administrators considering trying HPEL.

Throughout this article, "HPEL" will refer to the new log and trace system in WebSphere Application Server V8.5, and "Basic" will refer to the pre-existing log and trace system.

Introducing HPEL

HPEL is a new log and trace content storage and access system for WebSphere Application Server. HPEL stores log and trace content produced by the existing log and trace APIs, in particular the java.util.logging API.

HPEL addressses a number of common needs:

- **Log and trace performance**
  If you have worked with IBM support teams, you might have been asked to turn on tracing for certain components. This might have been problematic to do on your production systems, because enabling tracing could significantly affect the performance of your applications. HPEL log and trace is significantly faster than Basic log and trace, making tracing much less impactful to your application performance. Using the DayTrader benchmark and EJBContainer=fine trace specification, HPEL boosted overall benchmark performance by 33% compared to using the same trace specification with Basic log and trace (measured using WebSphere Application Server V8.0).
- **Log and trace entry extensibility**
  Being able to extend log and trace records can be very useful. For example, you might want to add a userID field to every log and trace record so that you can search your logs or traces later to find records with a particular userID value. There is no mechanism for doing this with the Basic log and trace system.
- **Automating log/trace analysis**
  As a WebSphere Application Server administrator, you might have had a need to automate the scan of SystemOut.log and SystemErr.log for significant events. This is non-trivial with the Basic log and trace system because log and trace entries can span multiple lines in the log/trace files. The SystemOut.log and SystemErr.log files also rolled (meaning that when they reached a certain size, they were renamed and truncated back to 0 length), adding further complexity to automation.
- **Application separation**
  To facilitate problem determination, you might want to separate the logs and traces for each application into a separate file. There is no mechanism with the Basic log and trace system to make that easy to do. If you had created your own log files per application (perhaps using Log4J), you might have found it cumbersome to correlate content in your log files with WebSphere Application Server log files, or to correlate log and trace content between applications when looking for patterns.
- **Request tracking**
  You might have wanted to gather the log and trace records that made up a single request to your server, where a request might have been served by multiple application server threads, or different application server JVMs (such as for web services requests). The Basic log and trace system does not

provide that capability. HPEL delivers this when used in combination with Cross Component Trace (XCT).

- **Remote access to logs/trace**
  The Basic log and trace system provides the ability to view chunks of the SystemOut.log, SystemErr.log, or trace.log in the administrative console (WebSphere Integrated Solutions Console). HPEL provides a greatly improved user interface capable of filtering and browsing content more easily. HPEL also provides the ability to export log and trace content from the administrative console without need for operating system access. HPEL log and trace content can also be remotely accessed via a JMX MBean interface, or via the HPEL API.
- **Access to z/OS® logs/trace**
  If you run your applications on the z/OS platform, you might have difficulty getting your application developers to look at the log and trace content when problems occur, as they are more comfortable in a Windows® or Linux® environment. HPEL lets z/OS and other operating system teams work with logs and trace in a consistent way.

If you have used previous versions of WebSphere Application Server, you're probably familiar with the main run time log and trace files. On distributed systems, these are the SystemOut.log, SystemErr.log, trace.log, and activity.log. HPEL replaces all of these files with a log repository, a trace repository, and an optional text log.

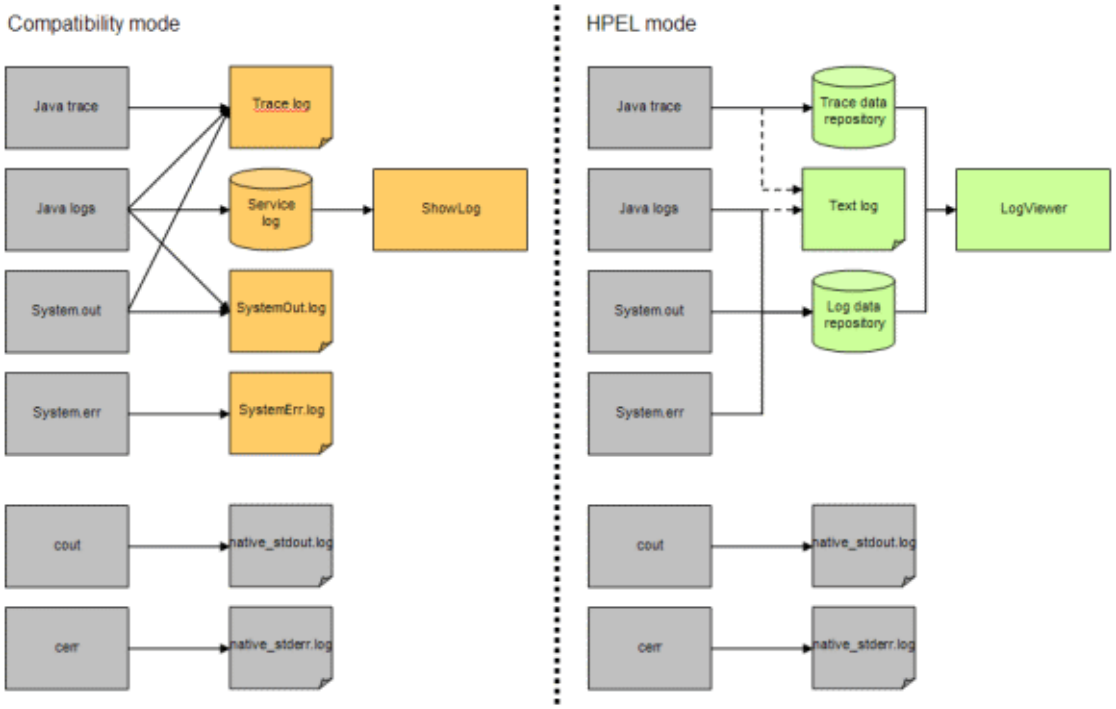
The Basic log and trace system stores log content in multiple places, including the SystemOut.log, the service log (activity.log), and the trace.log (whenever trace is enabled). System.err content is stored separately – in the SystemErr.log – making it challenging to correlate with content in the SystemOut.log. Trace content is stored in the trace.log. This makes the trace.log the primary log for problem determination when trace is enabled, and the SystemOut.log the primary log for problem determination when trace is not enabled.

In contrast, HPEL stores log, System.out, and System.err content in a log data repository, and trace content in a trace data repository (Figure 1).

To view the content from the log data repository and trace data repository, you can use the `logViewer` command line tool. The logViewer command provides access to all content in both repositories. It also enables live monitoring – similar to running `tail -f SystemOut.log` – and enables filtering so you can restrict what content it provides as output.

HPEL also provides a text log (which is optional, but enabled by default). The text log mirrors the content of the log data repository, but does so in plain text, with the same format you're used to from SystemOut.log. The text log can be enabled to additionally mirror the content of the trace data repository, though for performance reasons, you should avoid doing that on a production system. The text log is handy if you need a plain text log for any reason.

Figure 1. Comparison of where content is stored in Basic and HPEL modes



Configuration

The switch from Basic mode to HPEL mode is done on a server-by-server basis. The main **Troubleshooting > Logs and Trace > serverName** panel provides a button with which to switch each server to HPEL. Switching back to Basic is just as easy, but is done via the Change log and trace mode link on the same panel. In either case, you will need to restart the server whose log and trace mode you changed in order for changes to take effect.

The administrative console also provides HPEL configuration options for the log data repository, trace data repository, and text log, and lets you independently specify data retention and buffering options for each.

Be aware that HPEL can be configured for use on your deployment manager and node agent processes as well. Enabling HPEL on your node agent process is a good idea, as it enables you to use the Log Viewer in the deployment manager GUI to access the logs of any server on that node, even when those servers are stopped.

New in WebSphere Application Server V8.5

HPEL was first introduced in WebSphere Application Server V8.0. The initial design included the ability for the log and trace data repositories to have extensible records, but that capability was not surfaced to application developers or administrators.

In WebSphere Application Server V8.5, the application server uses this extensibility to store the application name in log and trace records created during requests to

Java™ EE applications. It also uses this capability to store the Cross Component Trace requestID, created when Cross Component Trace is enabled, in the log and trace records. Finally, an API is provided in V8.5 to enable you to add your own extensions to log and trace records.

To make use of these extension fields, the `logViewer` command line tool was extended to include an `-includeExtensions` option, enabling it to filter log and trace records by the content of extension fields.

Out of the box, WebSphere Application Server V8.5 tracks `appName` and `requestID` extensions, but you can feel free to add your own extensions to your log and trace entries using the com.ibm.websphere.logging.hpel.LogRecordContext API.

Let's take a practical look at how you might typically use HPEL.

Using logViewer command

When problems occur on your production application servers, it can be a time consuming task to discover and isolate the most important content in your log and trace files. Imagine that somewhere in your server's log files an error occurred on one of your systems.

Traditionally, you would have three options for finding problems in your log files:

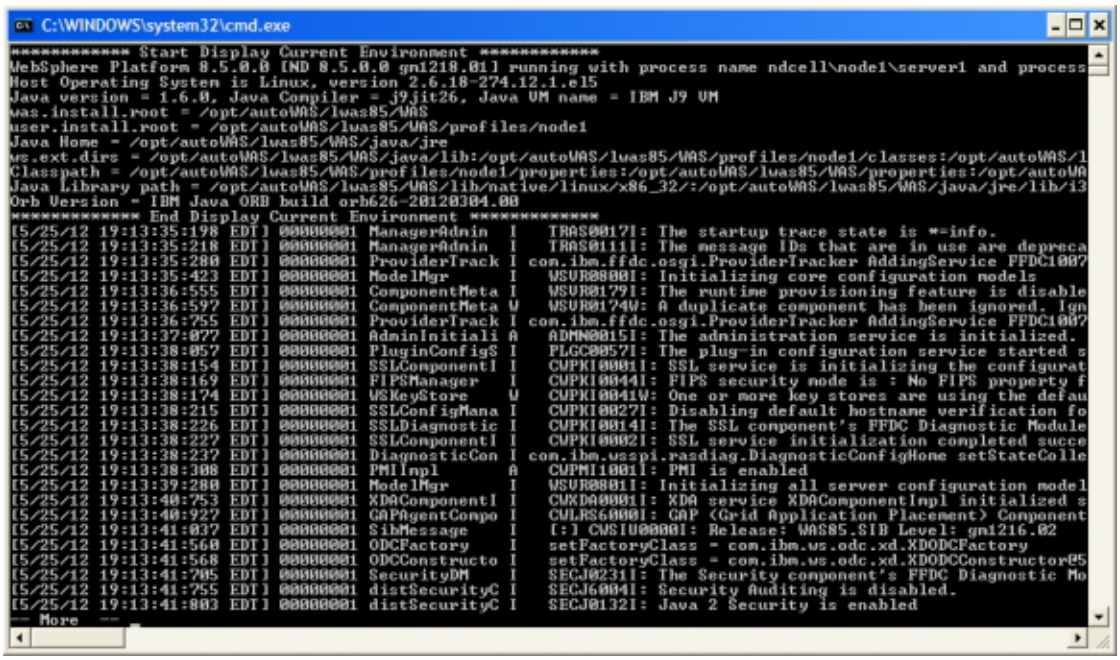
- Look at each log manually, perhaps grep-ing for " E " or " W " or "Error" or "Exception," and so on.
- Write scripts to do the above searches.
- Use tools to mine your logs for warnings, errors, and exceptions.

HPEL stores its log and trace data efficiently in binary repositories, which you can then access, filter, and format using the logViewer command line tool.

In its simplest form, you can output all content from your logs and trace using this simple command from your profile bin directory (output shown in Figure 2):

```
./logViewer.sh
```

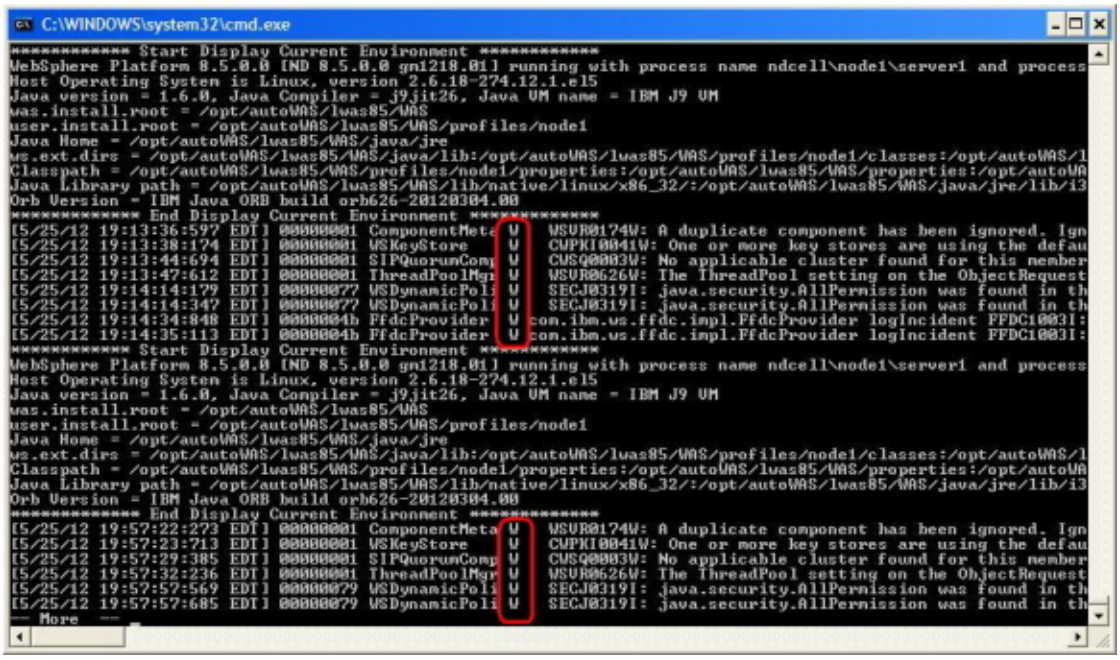
Figure 2. Output from logViewer.sh command



Let's try to do something more interesting though. To find any warning or error messages in your log file you can use this command (output shown in Figure 3):

```
./logViewer.sh -minLevel warning
```

Figure 3. Output from logViewer.sh command

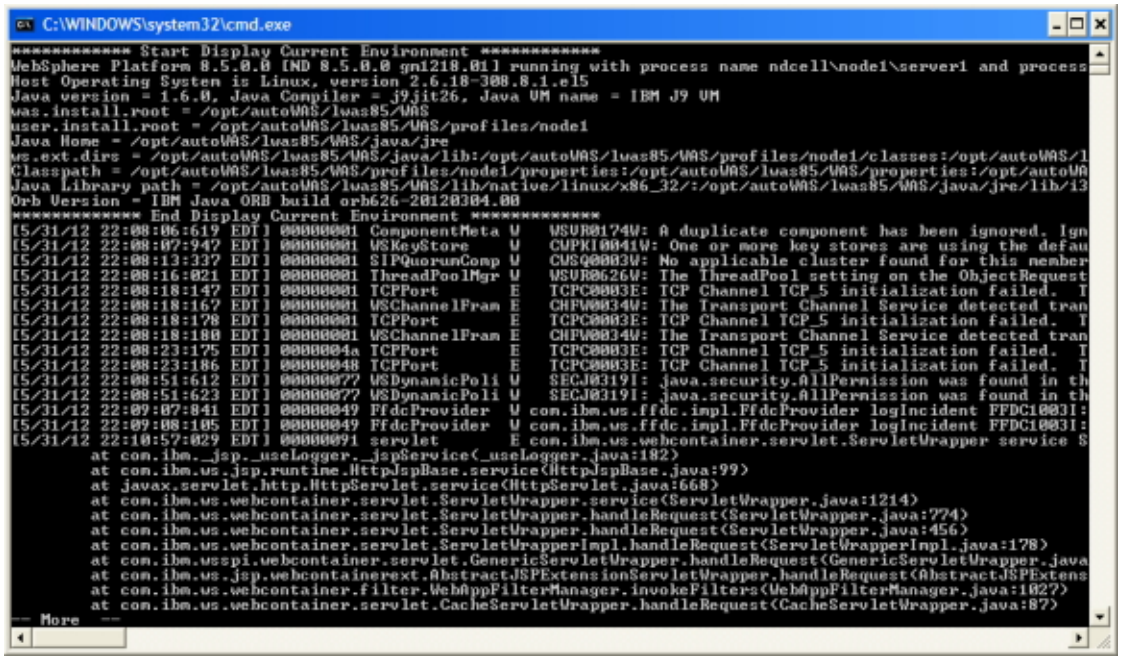


Notice in Figure 3 that the server was restarted, and by default the logViewer output includes the content from all ”instances” of the server. Importantly, logViewer never alters the repositories, it just provides a filtered, formatted view.

Now, let's further restrict your query to only see content since the last time you restarted your server. That's easy too, with the -latestInstance option (output shown in Figure 4):

```
./logViewer.sh -minLevel warning -latestInstance
```

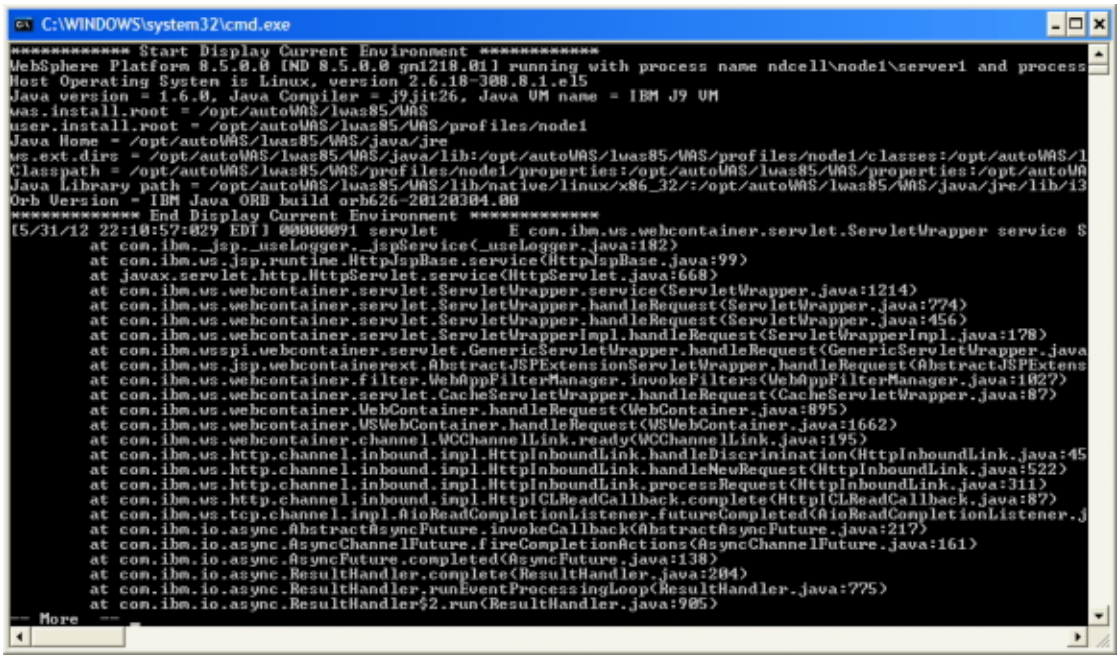
Figure 4. Output from logViewer.sh command



Maybe, you've even got a dozen applications running on the same WebSphere Application Server JVM -- and you'd like to check for warning or errors in your LemonadeStand application. Using WebSphere Application Server V8.5, you can filter your log and trace content by Java EE application name, leveraging the appName extension as follows (output shown in Figure 5):

```
./logViewer.sh -minLevel warning -latestInstance -includeExtensions
appName="LemonadeStand"
```

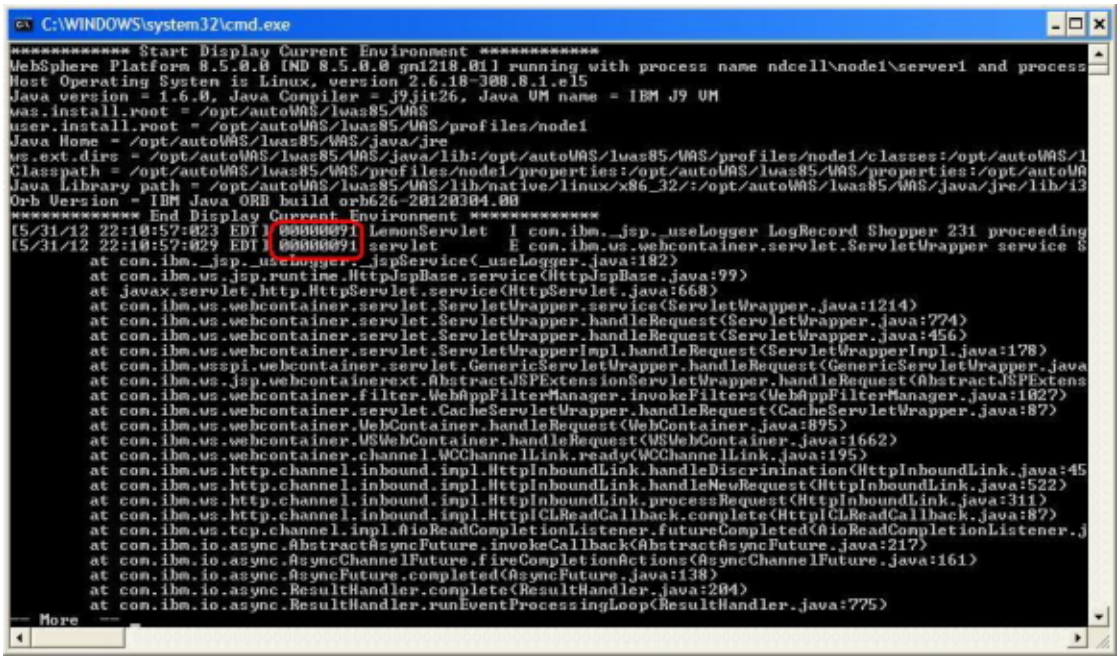
Figure 5. Output from logViewer.sh command



Now that you've found an interesting error in your logs, you might want to see what happened during the rest of the request on the same thread. This time, you can also eliminate any trace records by setting the minLevel option to info (output shown in Figure 6):

```
./logViewer.sh -startDate "05/31/12 22:10:40:000 EDT" -stopDate "05/31/12
22:10:58:000 EDT" -thread 91 -minLevel info
```

Figure 6. Output from logViewer.sh command

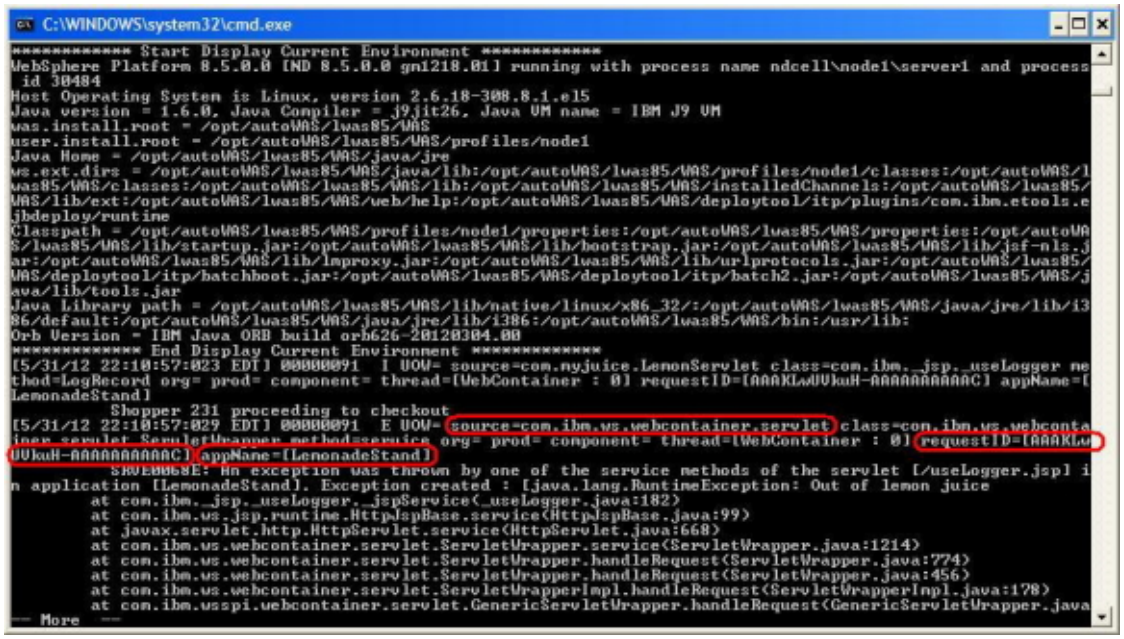


Using HPEL with Cross Component Trace

If you are using HPEL in combination with the new Cross Component Trace (XCT) capability introduced in WebSphere Application Server V8.5, and the error you are interested in occurred during an HTTP or SIBus request, you can filter your logs by (the XCT supplied) requestID. To see the requestIDs, rerun the previous query with the -format advanced option (output shown in Figure 7):

```
./logViewer.sh -startDate "05/31/12 22:10:40:000 EDT" -stopDate "05/31/12
22:10:58:000 EDT" -thread 91 -minLevel info -format advanced
```

Figure 7. Output from logViewer.sh command

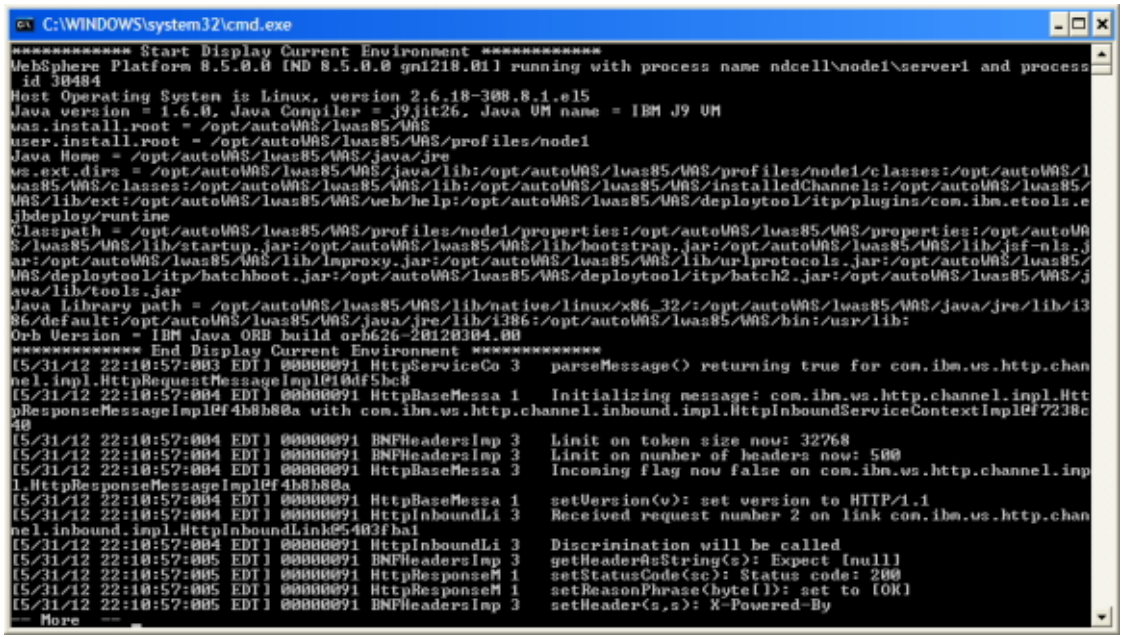


With the output in advanced format, you can see the requestID for that message. As another tip, viewing your logs in advanced format is also a great way to get the full logger name from the log entry (which appears as the 'source' field in log/trace records); this is helpful in case you want to enable more granular tracing for a particular logger. Similarly, you can see the name of the thread that logged/traced each entry in the advanced format.

Armed with the requestID of the request that had the error you are interested in, you can now query the repository to see other log and trace entries that are present in the same request. HPEL stores the requestID as an extension to log and trace records, so you use the -includeExtensions option, as follows (output shown in Figure 8):

```
./logViewer.sh -includeExtensions requestID=AAAKLwUVkuH-AAAAAAAAAAC
```

Figure 8. Output from logViewer.sh command



Share your log and trace data

Having isolated a problem in your log files, you might want to send the logs for your LemonadeStand application to a teammate. You can store your output in a log in plain text using the -outLog option:

```
./logViewer.sh -includeExtensions appName="LemonadeStand" -outLog /tmp/myLogs.log
```

Alternatively, you could store your filtered output as a new HPEL repository, using the `extractToNewRepository` option, so your colleagues can use the logViewer command on their system to continue to manipulate the data:

```
./logViewer.sh -includeExtensions appName="LemonadeStand" -extractToNewRepository /mySharedDrive/myLogs
```

Upon receiving word of your great work, your colleague could then access what you stored in the log. To do so, they would specify the location of the repository using the `repositoryDir` option:

```
./logViewer.sh -repositoryDir /mySharedDrive/myLogs
```

View log or trace data in real time

When developing or troubleshooting applications, it's often helpful to be able to monitor the log output in real time. Perhaps you are aware of a problem and are trying to get it to reoccur, or you want to watch for a specific event as part of testing

your application. At one point or another, you've probably found a need to fire up a shell window and execute a `tail -f SystemOut.log` to monitor your log output.

HPEL can make this type of task easy by enabling you to monitor a log repository for new events without having to monitor multiple log files or worry about files rolling over, interfering with your ability to tail them. Try it out by simply invoking the HPEL logViewer tool with the `-monitor` option:

```
./logViewer.sh -monitor
```

A key benefit with using HPEL's monitor option is that it works across platforms, providing you a consistent option for monitoring new log and trace events, regardless of your hardware or operating system. This includes the z/OS® platform, so it's possible to easily view or work with your application logs on z/OS without needing to be a z/OS system administrator or have access to log spools.

The monitor feature can be combined with filtering options to watch for specific events. Imagine you need to monitor a verbose logging application for critical error messages that pertain to a specific username that is part of a message. You could do this with something like the following to see all new severe or fatal log events that contain the string “user33” as part of their message, without the need for grep or other tooling:

```
./logViewer.sh -monitor -minLevel SEVERE -message “*user33*”
```

As you might expect, it is possible to combine multiple filter options to further fine tune what log events you want displayed. For example, you could further limit the records displayed to those generated by the LemonadeStand application as follows:

```
./logViewer.sh -monitor -minLevel SEVERE -message “*user33*” -
includeExtensions appName=”LemonadeStand”
```

By combining options for the logViewer tool, administrators can easily view all or only the specific log data that is required directly from the HPEL repository. LogViewer provides a consistent and powerful tool for viewing WebSphere Application Server log and trace data regardless of platform. It does not require that any special tooling be installed or administrators to copy log files to their workstations.

Now that you've seen how easy it can be to use HPEL to view live log data and filter that output from a command line, lets look at how easy it is to view log data from within the WebSphere Application Server administrative console.

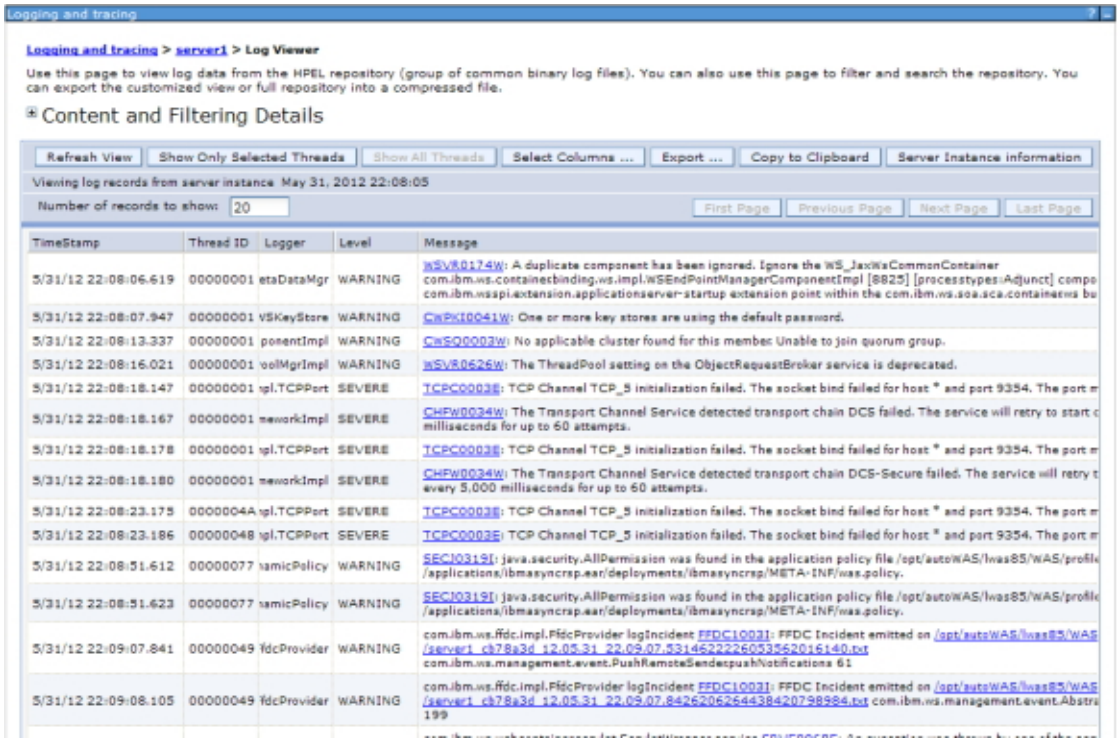
View log or trace data from the admin console

Another common scenario that many system administrators may have faced is the need to ensure limited operating system access to key production application servers, while still allowing application support teams or development groups access

to the production application's logs. WebSphere Application Server now provides an improved web-based log viewing and export utility via the administrative console when HPEL is enabled.

While Basic mode log and trace has for some time provided simple log viewing capabilities, HPEL provides an improved console log viewing experience. The HPEL Log Viewer (not to be confused with the logViewer command line tool) can be used to view HPEL log and trace repository data by anyone authorized to access the administrative console. An application support specialist with Monitor role access to the administrative console can view log data without needing OS level access to the production servers or having to wait for their system administrator to collect the log files on their behalf.

Figure 9. Administrative Console view of log contents

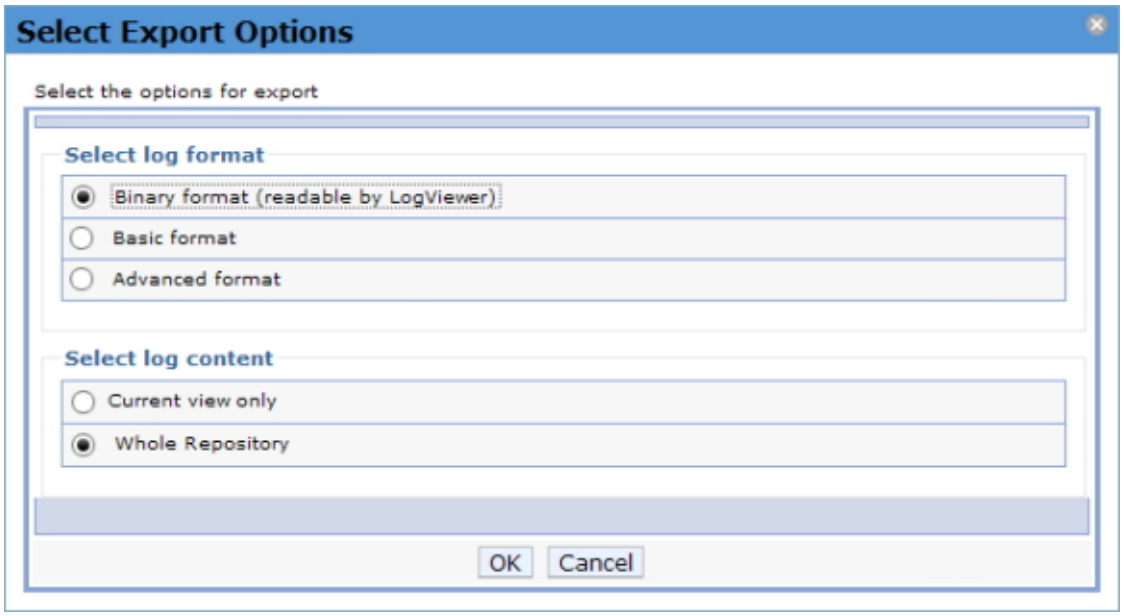


The console's Log Viewer provides all the same powerful filtering options found in the command line logViewer tool. To view log and trace data via the administrative console, navigate to **Troubleshooting > Logs and trace** and select your HPEL enabled application server from the list, then choose the **View HPEL logs and trace** link on the Logging and tracing panel.

The full log repository (or even a filtered subset of the repository) can be exported to a .zip file for local analysis, if required, directly from the administrative console's Log Viewer. Simply select the export button at the top of the Log Viewer. You can choose to export the entire repository or just your current (filtered selection) to a new binary HPEL repository. Anyone you share the repository with can use the logViewer command to manipulate the file later. If preferred, you can export the

entire repository or the current selection to a text file in basic or advanced format as well. The logViewer command-line tool can read HPEL repositories from inside an exported zip file -- no unzipping needed.

Figure 10. Administrative Console export log dialog



Conclusion

HPEL utilizes a new way to store log and trace data for improved performance and extensibility. At the same time, HPEL maintains compatibility with your existing log manipulation scripts by providing the same data formats you are already familiar with from WebSphere Application Server logs. New command line and administrative console tools expedite the tasks of log and trace monitoring and filtering. In addition to the tools provided, tool providers can utilize the new HPEL Java API, which provides access to HPEL repositories locally as well as remotely, and output results in basic or advanced text formats, or even store them in a new HPEL repository (see Resources).

Switching to and from HPEL in WebSphere Application Server can be done quickly in the administrative console and requires a server restart to take effect. This ability to easily switch between logging modes means you can try out HPEL and switch back if you find any need for the Basic mode. IBM support teams are able to work with HPEL repositories – whether exported from your administrative console, output from the logViewer command, or simply zipped up from your server logs directory.

Resources

- WebSphere Application Server V8.5 product information
- WebSphere Application Server V8.5 Information Center
- High Performance Extensible Logging (HPEL)
- IBM Education Assistant - High Performance Extensible Logging
- Javadoc – com.ibm.websphere.logging.hpel.reader
- Javadoc – com.ibm.websphere.logging.hpel.writer
- Cross Component Trace (XCT)
- WebSphere Application Server V8.5 trial version
- IBM developerWorks WebSphere

About the authors

Don Bourne

Don Bourne is the WebSphere Serviceability Architect. Don joined IBM in 1996 and helped to build what is now known as WebSphere Commerce. Don has worked in various roles in server development, performance measurement and improvement, and problem determination. Don joined the WebSphere Application Server team in 2003, and has focused on making the application server, and stack products more easily serviceable.

Igor Belyi

Igor Belyi is a developer in the WebSphere Serviceability group. Igor joined IBM in 1999 through acquisition of Transarc Corporation. Igor worked as a developer on Encina/TxSeries/CICS middleware product, Dump Anaylzer framework, IBM Support Assistant, and Knowledge Centered Support. Igor joined WebSphere Serviceabiliity team in 2006.

Scott Highbarger

Scott Highbarger is an Advisory Software Engineer in the IBM Software Group.

The importance of embracing the performance lifecycle

Incorporating performance considerations throughout the development process

Keri-Anne Lounsbury (kerianne@ca.ibm.com)
Delivery Manager
IBM

Skill Level: Intermediate

Date: 01 Aug 2012

Kevin Yu (kevinyu@ca.ibm.com)
Senior Certified Consulting IT Specialist
IBM

This overview of the fundamental concepts behind the performance lifecycle will help you understand the activities and involvements needed from various stakeholders to ensure a successful implementation.

Introduction

Performance is an area in which IT teams often seek consultative advice, whether it be for general guidance or to address specific critical performance issues that threaten the stability of an application or website. While either is a clear and valid reason to discuss website performance, what might be less clear is exactly what is meant by “performance,” what are the things that impact it, who needs to be involved to improve it, and perhaps most important: why it needs to be an integral part of the development process.

This article outlines the important ideas behind the **performance lifecycle** to help you prepare for what is often one of the most challenging, critical, and neglected elements of web project development, both from a project management standpoint and an execution standpoint. Questions to be addressed include:

- Why do both the business sponsors and IT sponsors need to care about and understand performance fundamentals?
- What does “performance” encompass?

- Why is it necessary to have a business plan in order to achieve a high-performing website?
- What is the performance lifecycle?
- How should the business and IT teams collaborate to build an effective performance strategy?

The importance of performance

Performance has a direct impact on business objectives, and vice versa. For example, the highest priority for nearly all retailers, is to drive revenue and turn a profit. To do this, one retail business goal might be to maintain a competitive edge with exciting and unique customer experiences. This can include offering products and services to the consumer in a meaningful way that encourages them to make a purchase and return for more, or providing an experience that illustrates the value of the goods and services offered and lets customers communicate with like-minded individuals, and so on, all to maintain an advantage over a competitor who is just a click away.

To achieve these critical success factors, an underlying requirement is that the website must provide a reliable user experience, giving shoppers what they want with very rapid response times. If you consider that all of this must be done in a profitable way, a key metric becomes the return on investment of the hardware and software considered in proportion to the amount of revenue that will be gained; in performance terms, we talk about the amount of the hardware and software as aspects of the capacity plan.

So, what if you do not consider the performance of a website, either before launch or prior to other significant events on the site? Could the effects really be that detrimental? The answer is yes, and emphatically so. Here are just a few possible outcomes of not adequately considering performance in your development cycle:

- **Website failures during peak times**. In the brick and mortar world, this is akin to not being able to open the front doors of your store. This leads to:
    - Lost revenue.
    - Negative customer experience, which tarnishes your brand and encourages your customers to visit competitors.
    - Potential legal issues and penalties. Examples might include affiliates and partners to whom you have commitments, which you are unable to meet with an unstable or unavailable site, or scenarios where there are time commitments for completing a transaction (especially important in financial transactions).
    - Increased costs for your customer support organization, as your call center becomes busy during times of online crises.
- **Website slow downs**, which lead to:
    - Perception of the site being unavailable. All of the negative aspects of an unavailable site apply here. With the online world always getting faster and

competition constantly growing, it takes much less time than you might think for users to get impatient with your site.
    - Compounded performance problems. Perception that a user request has not gone through might cause users to try again, creating frustration, bottlenecks, and additional complexities for handling duplicate transactions.
- **Patchwork solutions by adding more hardware and software** than originally anticipated to mask website inefficiencies:
    - While this may work in the short term, the costs of such a strategy take away from profitability, and could tarnish the reputation of the development or IT team responsible for the website.
    - This strategy could drain funds from other important projects.

The story is not all doom and gloom though. While the intent is to clearly show that by ignoring performance the negative impacts can be significant, the remainder of this article presents ideas to help you begin implementing a high performing site that meets critical business needs at all times, including the most important times for your business.

The impact of performance

Application performance is often equated to **responsiveness**. Since speed is subjective, its definition varies depending on the audience and organization. A retail company might consider its website performance to be fast if it can serve catalog pages and place orders in a few seconds. In contrast, a stock trading application measures speed in milliseconds. As a user, you might tolerate a few seconds to perform a banking transaction online, but if a shopping site is slow, a viable option might simply be to navigate away to an alternative site.

The retail example highlights the importance of application performance and how it connects to a company's bottom line, as every minute on a busy day could translate to significant sales won or lost. Performance, therefore, is not merely a benefit for user experience. It contributes to a site's reputation, boosts customer loyalty, and can have significant impact on site revenue. Ultimately, performance is not only about optimization and speed. It is about contributing to the bottom line of a business.

Another aspect of performance that varies greatly across projects is the execution of the **performance strategy**, which can vary by organization. Too frequently, the idea of performance testing is equated with adding a test phase, believing that performance risks can be covered by simply running a performance test right before launch. While this is better than assuming performance will come naturally or be taken care of by the support teams, the reality is that to ensure a site will perform, much more than a few tests will be required. A test phase might discover bottlenecks in the application, but the fix might require a solution that goes back to the fundamentals of application design and architecture. Considering performance only at the end of development leads to not being able to address issues on time

or having to patch and get by. In the long term, the "band-aid" mode leads to an increasingly complex application that becomes more difficult to manage and optimize. The point of the performance lifecycle is about considering performance in all aspects of the project, not as only a one-time event.

The performance lifecycle

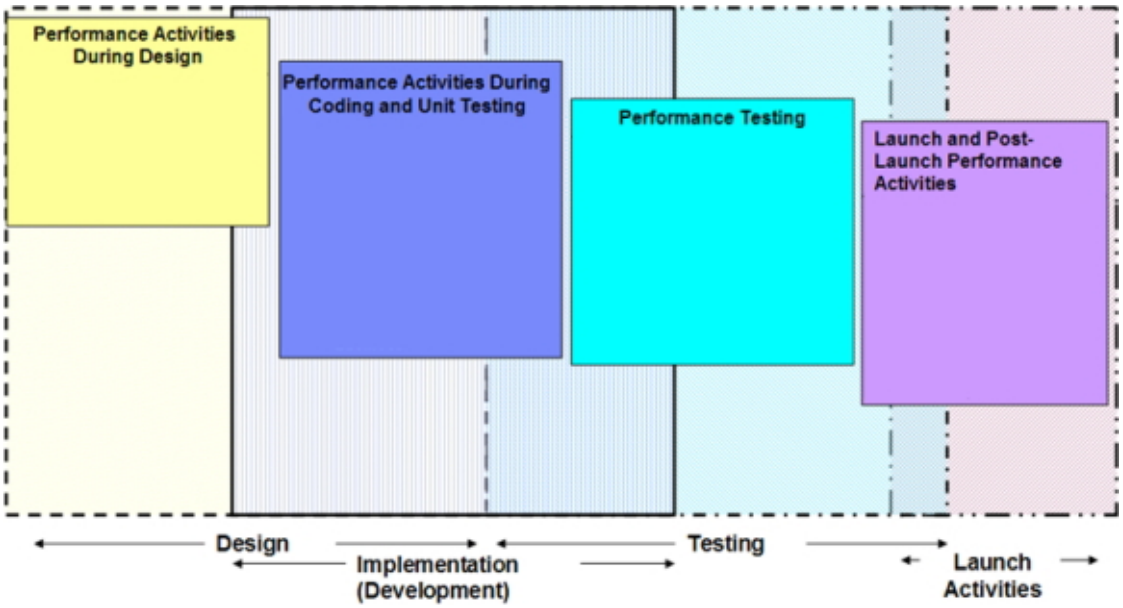
Traditional project plans leave significant performance activities unmanaged for the majority of the project. Performance is considered in the plan as a test phase near the end of the cycle, when there is usually insufficient time and resources to achieve performance objectives because problems discovered so late can be too severe to be corrected before the rapidly approaching deployment date. Furthermore, because development is already finished at this stage, solutions are often provided as afterthoughts or workarounds, which can complicate the application logic for future development. In some circumstances, the root cause of bad performance could be due to system architecture, which would be next to impossible to address this late in the project cycle.

The premise of the performance lifecycle is that performance is considered throughout the entire project lifecycle. Adapting the performance lifecycle approach shifts performance from a point in time activity to a parallel and equally important track alongside a project’s development activities. This raises the visibility of performance throughout the project and its deliverables become key aspects of the project plan. Additionally, achieving pre-defined performance criteria becomes part of the website or application launch approval process. The key performance metrics are continually addressed and tracked against targets.

Performance lifecycle activities

Figure 1 shows the placement and timing of performance-related activities during the traditional design, implementation, testing, and launch stages of the project lifecycle.

Figure 1. Performance lifecycle at a glance



Activities for each phase of the performance lifecycle include:

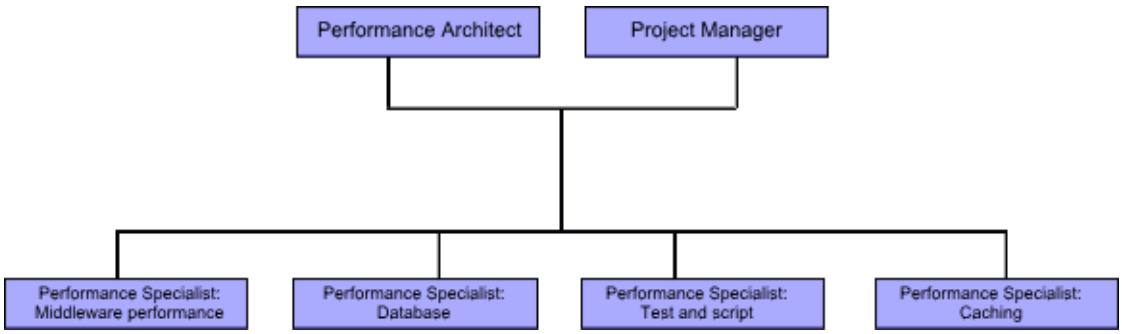
- **Design**
  - Review and validate non-functinoal requirements with IT and business stakeholders.
  - Assist with application flow design.
  - Educate developers on performance.
  - Review component design focused on performance.
  - Prepare system maintenance strategies.
  - Identify key performance indicators (KPI) for monitoring
- **Coding and unit testing**
  - Developer-performed application profiling (phase 1 of performance validation).
  - Identify initial cache strategy.
  - Implement system maintenance strategies.
  - Build performance verification environment (including data).
  - Finalize performance testing plan.
  - Plan priorities and risk mitigation with business stakeholders.
  - Instrument code to track KPI.
- **Performance testing**
  - Create scripts and test data, validate environment, and so on.
  - Multiple-staged test approach applied.
  - Test single system and scale-up.
  - Run endurance tests.
  - Test availability at end of this phase.
  - Plan priorities and risk mitigation with business stakeholders.
  - Refine KPI monitoring strategy.
- **Launch and post-launch**

- Migrate tuned settings to production.
- Monitor and troubleshoot production system.
- Analyze production access logs, advise on script rework.
- Performance test initial round of post-launch fixes.
- Ongoing interlock with business stakeholders to understand future plans.

Performance lifecycle roles

As your project plan adopts the performance lifecycle, your resource plan also needs to reflect the continued involvement of performance resources. Key performance roles are shown in Figure 2. The performance architect, project manager, and product specialists are often engaged for the entire lifecycle, with additional test specialists and analysts added during the test phase.

Figure 2. Project planning key performance roles



Responsibilities for each of these key roles include:

- **Performance architect**
  - Performance team leader
  - Overall leadership in performance aspects of solution architecture and design
  - Ownership of performance-related non-functional requirements
  - Leads performance verification activities
- **Project manager**
  - Overall coordination of performance engagement activities
  - Reports results
  - Drives issue resolution
  - Links performance with project manager for overall project
- **Performance specialist: Middleware performance**
  - Code profiling analysis and recommendations
  - Analyze results, debug and tune middleware applications
  - Guidance on performance test design
- **Performance specialist: Database**
  - Performance analysis and recommendations for SQL, database indexes, configuration, and so on
- **Performance specialist: Test and script**

    - Develop and maintain workload, scripts, test data reporting
- **Performance specialist: Caching**
    - Cache design and implementation

Adopting the performance lifecycle requires project, resource, and process changes to give performance a level of visibility that is tracked and accounted for throughout the project. With this level of change, successful implementation also requires a cultural change of the organization's mindset on performance. Furthermore, as the priority of performance is often lowered in lieu of deliverables and timeline, sponsorship and support must be present from executives and the senior leadership team to make this a reality.

Adopting the performance lifecycle

To help you plan for adopting the performance lifecycle into your business, a sampling of actions that apply at various stages of a project are presented below. Incorporating these actions will help you begin transitioning your project culture away from the traditional approach of including performance as a conclusion to the project testing phase, and toward adopting performance considerations at every project stage.

Requirements definition

Performance priorities:

- Establish key performance and business objectives for your site, such as:
    - What do your users expect or require when transacting with your site?
    - What are the business and transactional objectives for the site in the next 12-24 months?
- Identify requirements that will have a significant impact on capacity or utilization of system resources. For items that could have a negative impact on performance, negotiate to find out if these are truly must-have items, or if similar objectives can be met with a different (and less performance intensive) approach.
- **Table 1. Requirements definition example**

Not very performance friendly	Better
Display the entire storage catalog in real time on single page.	Present catalog data to users in increments as needed.

Risks from not considering performance during requirements definition:

- Introduction of features that could have negative performance impact without much business benefit.
- Add delay during implementation phases to triage and optimize the performance of the feature.

- Open door for a “reactive” approach to performance and capacity considerations that can take resources and time away from other projects, rather than a “proactive” approach.

Design

Performance priorities:

- Architect a solution that has a good balance of features and performance considerations.
- Design and build performance test environment. Begin to deploy tools that will be required to support your performance strategy.
- Identify architecture limitations that could challenge a non-functional requirement.
- **Table 2. Design example**

Not very performance friendly	Better
Inventory information is retrieved via a web service request against a geographically remote service provider.	Locally cached inventory information is refreshed automatically when remote data changes.

Risks from not considering performance during design:

- Increased possibility of features consuming higher resource utilization than projected over the capacity plan estimate.
- Application performance and user experience can be adversely affected by third party application interfaces.
- Lack of proper test environment can hamper performance testing capabilities and put test results into question.

Coding and unit testing

Performance priorities:

- Measure the response time of business critical steps in the development environment.
- Profile code execution to identify patterns against best practices.
- **Table 3. Coding and unit testing example**

Not very performance friendly	Better
Iterate through items in the shopping cart to calculate applicable promotion.	Single calculation of applicable promotion against a list if items.

Risks from not considering performance during coding and unit test:

- Potential of discovering inefficient and resource intensive code late in development, which can often jeopardize key milestones.
- Efforts required for identification and triage of performance issues increase the later they are found. Environments become more complex and are typically

under increasing amounts of change control as code gets closer to production, which increases the effort and time required to get to resolution.

Performance testing

Performance priorities:

- Start with simple, common test cases and gradually go to more complex scenarios and environment configurations.
- Measure the performance of the application as load scales up to projected peak.
- Identify and resolve the root causes of resource constraints and application bottlenecks.
- **Table 4. Performance testing example**

Not root cause	Better
Raised maximum JVM heap size because JVM crashed due to OutOfMemory condition in a duration test over 6 hours.	Find a memory leak that was identified by examining the heap dump after the OutOfMemory crash.

Risks from not executing a well defined performance testing methodology:

- Serious business revenue impact as application breaks down under peak load that was not tested prior to launch.
- Increased deployment and system maintenance costs if additional hardware is viewed as the fix for performance issues.
- Very difficult performance triage required in production environment if performance issues are not caught by testing, and instead are directly impacting users.

Launch and post launch

Performance priorities:

- Instrument and monitor performance indicators in the production environment to enable the team to proactively identify and address potential issues before users are affected.
- Foster communication between marketing, performance, and operation teams to be better prepared for promotional events.
- Use the data captured from production to optimize planning for future marketing and promotional activities.
- **Table 5. Launch and post launch**

Not very performance friendly	Better
Marketing team sent out an email promotion campaign to 2M registered users with a search page URL as landing page.	Landing page is a static page, cached on the edge. The performance team also tested the projected increase in traffic for the event.

Risks:

- Bad user experiences lead to lost customers.
- Capacity below actual demand, which degrades application performance and user experiences.

The above actions are a sampling of ways you can begin to bring performance considerations into your project, as well as risks to projects that do not adopt lifecycle approach to performance.

Conclusion

The performance characteristics of your site are not just a technical consideration, they are also a business consideration. By adopting a performance lifecycle approach into project implementations, you can balance the business objectives of the site with the appropriate technical implementation to withstand the traffic and pressures of your site.

The best examples of adopting the performance lifecycle begin in the requirements gathering phase and continue through monitoring of actual site performance in production. Design phases include reviews and evaluation of implementation best practices such as code optimization, caching opportunities and the latest performance enhancing technologies. Testing phases include not only functional validation but also validation of the user experience under peak load. By executing these steps, problems can be identified earlier in the development cycle which can improve the efficiency of development and help minimize unnecessary costs.

Finally, in a business climate that constantly evolves to create new and exciting features to increase revenue and remain competitive, your team should embrace the performance lifecycle as part of each release plan.

Resources

- WebSphere Application Server Performance
- Blog: Application Performance Management
- Information Center: Caching strategy
- Information Center: Example methodology for performance testing phase of a WebSphere Commerce implementation
- Best practices and case studies: High Performance On Demand Solutions (HiPODS)
- Book: Performance Analysis for Java Websites
- IBM developerWorks WebSphere

About the authors

Keri-Anne Lounsbury

Keri-Anne Lounsbury is a Delivery Manager in IBM Software Group. Keri-Anne has 15 years of experience in the IT industry with a deep focus on the e-commerce space. Her experiences include leading customer and IBM teams in the delivery of complex WebSphere Commerce projects, as well as helping customers resolve critical issues and prepare sites for peak performance.

Kevin Yu

Kevin Yu is an application performance specialist with fourteen years' experience in the IT industry. Kevin has worked on application server, enterprise commerce and database products. He has led multi-disciplinary teams to solve critical customer situations and guided many to achieve new records in transaction volumes and sales on their busiest online shopping days of the year.

Performance testing and analysis with WebSphere Application Server

David Hare
Advisory Software Engineer
IBM

Skill Level: Intermediate

Date: 01 Aug 2012

IBM® WebSphere® Application Server supports an ever-growing range of applications, each with their own unique set of features, requirements, and services. Proper performance testing and analysis on each of these applications is essential to ensure they are performing at their maximum potential. This article provides guidance on some best practices on how to build a performance test, compare results across application or environment changes, and how to identify bottlenecks using freely available tools from IBM. The methodologies described here apply to all versions of WebSphere Application Server, including the newly released WebSphere Application Server V8.5.

Introduction

Can you relate to any of these statements:

- *Currently we're not doing any performance testing. We'd like to but we're just not sure where to even begin.*
- *Our application was running fine, but after the development team sent us an updated version to deploy, we're seeing much higher CPU usage on the server. Where is the high CPU usage is coming from?*
- *We migrated our application from version 2.0 to 3.0 and now response times are three times longer. What is causing these delays?*
- *Our application needs to support 40% more users in the next three months. How can we prepare for that other than simply adding more machines to the cluster?*

These statements represent very common scenarios, so if any of these sound familiar, you're not alone. The goal of this article is to address these types of situations by suggesting some best practices with regard to basic procedures for conducting tests and useful tools that are available. At a high level, the major topics discussed here will help you:

- Write useful performance test cases.
- Drive varying amounts of load to stress low and high utilization.
- Record key performance and system metrics.
- Conduct performance testing in parallel with application development cycles.
- Use tools like IBM Health Center to find performance bottlenecks.
- Work with development teams to fix bottlenecks and re-measure.

The big deal about performance testing

At some point along the way, performance testing got penciled in as something that is done just before rolling into production. It's frequently a very minimal effort with not enough time allotted to identify and fix real problems that eventually will show up in the production environment. The universally recommended approach to proper performance testing is to implement the performance lifecycle, in which performance testing is scheduled as part of the development work, iteratively testing as new features are integrated. This enables bottlenecks to be identified and resolved with plenty of time left in the release cycle before everything is rolled into production.

Another benefit of proper performance testing is the opportunity to tune the environment (operating system, JVM, application server, application, database, and so on) for maximum performance. Only through proper performance testing can tuning parameters be assessed to determine if they are providing any value. Many users set JVM heap sizes based on developer recommendations and don't tune anything else because it isn't considered necessary. You might be surprised to learn that it might be possible to cut the amount of hardware needed for an un-tuned environment in half just by conducting some simple tuning steps. This article proves the point.

With some simple tuning procedures, the DayTrader performance benchmark application can handle over 2x the load as the un-tuned environment. This means the same number of users could potentially be supported with half the available hardware resources. Think about the costs that could save.

In addition to iterative testing throughout the development cycle and the benefit of testing for tuning purposes, the other major advantage of extensive performance testing is the ability to compare results across application and environment changes. Real performance testing records key metrics (discussed later) that enable administrators to gain insight into problems that might be arising. To go back to one of the common comments above, many users aren't prepared to figure out where a problem is coming from when they migrate to a newer version of an application because they never did proper performance testing or recorded key system metrics for the earlier version. Without it, a test server with the earlier application version will likely need to be setup as a comparison point. Having this type of data makes analysis of where the regression is coming from much easier to find.

Best practices for proper performance testing

There are two fundamental best practices for proper performance testing that can be summed as follows:

- **Vary the number of users (or client load).** A production environment typically has a varying number of active users throughout the day. Quality testing ensures the application performs well under small loads and peak (for example, Black Friday) loads. This might mean changing around the “think” time in between requests and changing around the active number of users hitting the application. One of the best ways to do this is to perform a test with 1 active user, 2 users, 4 users, 8 users, and so on. You will see this in practice later.
- **Record key system and performance metrics.** There are several important metrics that should be recorded for all scenarios. For each performance run, the most important metrics to record are:
    - Throughput (requests per second)
    - Response times
    - Application server machine CPU utilization %
    - Other machine CPU utilization % (web server, load driver, database, as applicable)

The throughput and response time metrics can be seen in whichever load driving tool is being used. The CPU utilization, memory utilization, disk I/O, and network traffic metrics can be seen with tools like vmstat or nmon on Linux® or AIX®, or Task Manager on Windows®. In addition to the above metrics, it is also important to record all system level information. This includes operating system level, number of active cores, how much memory (RAM) is available, the “Java™ version” output, the WebSphere Application Server version information, and all tuning that has been applied. Recording all of these metrics will enable you to quickly compare scenarios, even if the scenarios being compared were tested two years apart.

Many users don't have the hardware available to replicate their production environment with a testing environment of the same size. In these cases, the recommended approach is to scale the performance test based on the resources that are available. For example, assume a production environment consists of ten physical machines, each running two instances of WebSphere Application Server. If only one physical machine is available for performance work, this machine could be setup as identical as possible to the production machines, and the load driven in the performance test would be roughly 10% of the expected production workload. Eventually the performance test should be ramped up to replicate the full production environment. This way, other processes such as the database and LDAP are load tested as well.

Test cases and load drivers

The very first step in conducting performance tests and finding application bottlenecks is to write useful test cases. Results and analysis are only as good as

the test case that was used to produce them, so this step should not be taken lightly. Stressing application code paths that users only hit less than 10% of the time is not nearly as beneficial as stressing code paths that all users will hit. Focus on the most popular code paths first, and build your tests down to the lesser utilized code paths later. Spend a lot of time here really investing in a quality test case that emulates actual user traffic. This developerWorks article is a great resource for getting started in writing performance tests cases.

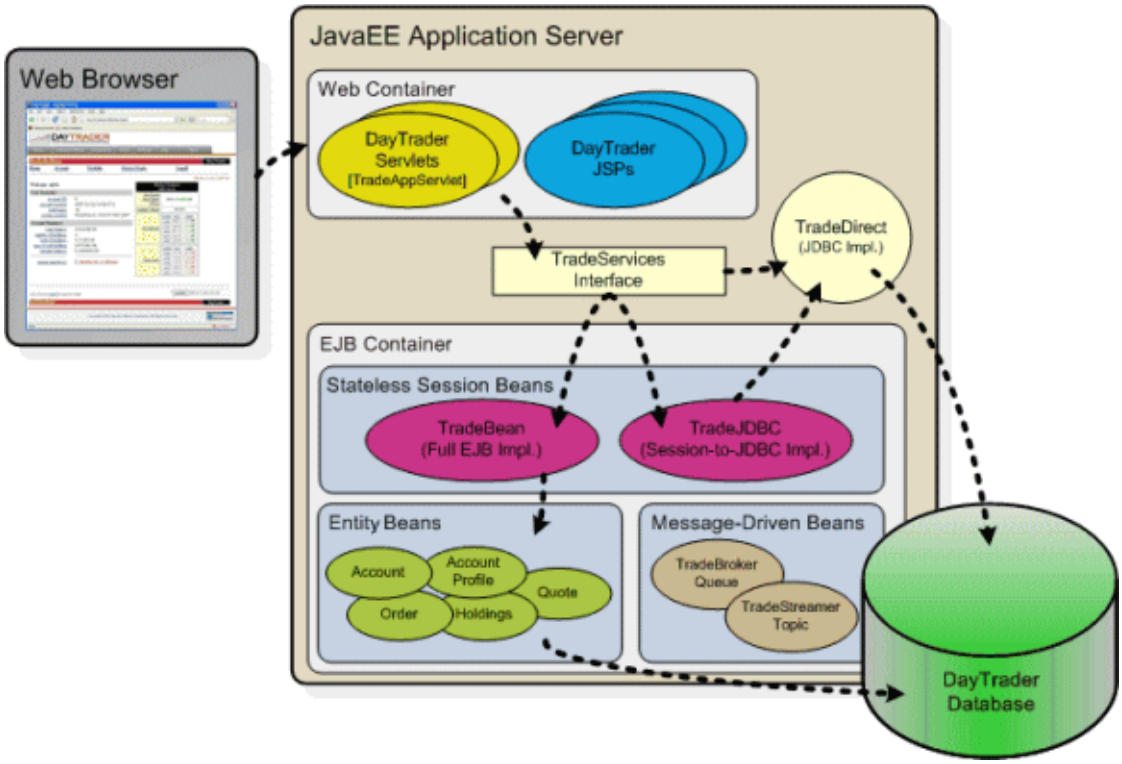
After mapping out a test case concept, you'll need to put it into practice with a performance load driving tool. There are many load drivers available to choose from, including IBM Rational Performance Tester and Apache's open source Jmeter. We'll refer to the latter in this article.

An example with DayTrader

Chances are good that if you've read any performance articles on developerWorks before, you've already heard of the “Trade” or “DayTrader” benchmark. The Apache DayTrader Performance Benchmark Sample application simulates a simple stock trading application that lets users login/logout, view their portfolio, look up stock quotes, buy and sell stock shares, and manage account information. DayTrader not only serves as an excellent application for functional testing, but it also provides a standard set of workloads for characterizing and measuring application server and component level performance.

DayTrader is built on a core set of Java EE technologies that include Java servlets and JavaServer™ Pages (JSPs) for the presentation layer, and Java database connectivity (JDBC), Java Message Service (JMS), Enterprise JavaBeans (EJBs) and message-driven beans (MDBs) for the back end business logic and persistence layer. Figure 1 shows a high-level overview of the application architecture.

Figure 1. DayTrader application overview



IBM has published a sample DayTrader package for download which includes the DayTrader application and the required deployment descriptors that you can install on WebSphere Application Server V7.0 or newer releases.

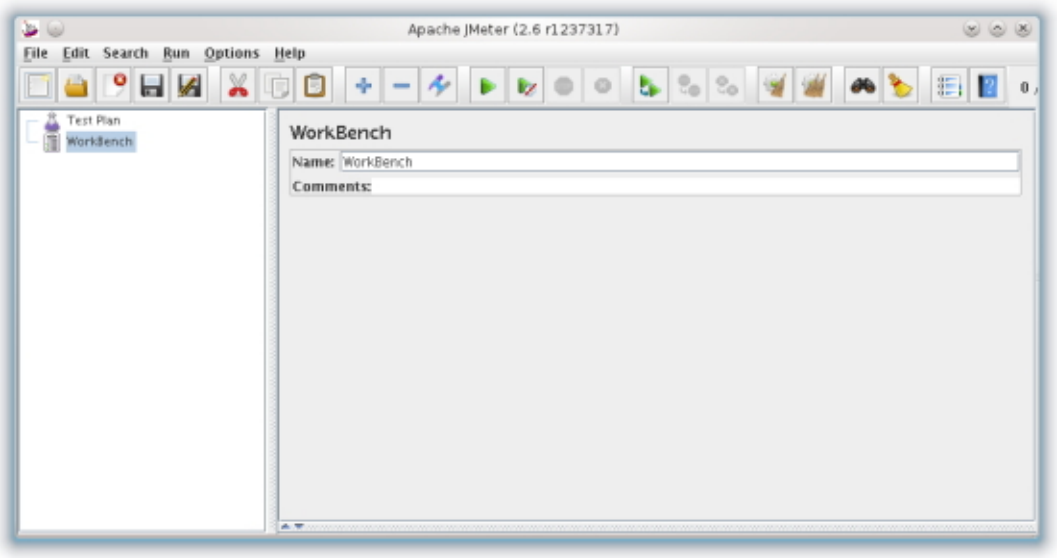
In this example, the DayTrader sample application was deployed to a base instance of WebSphere Application Server V8.5. One of the neat features of DayTrader is the **TradeScenarioServlet** link under the **Configuration** tab. This links to a servlet that emulates a population of web users by randomly generating a specific DayTrader operation for each user that accesses the servlet. (For example, one user might view their portfolio, one might perform a stock buy operation, one might look up a stock quote, and so on.) This ensures each of the main operations in DayTrader are executed during the test, and over time, because it's random, each operation should be executed roughly the same number of times. There are numerous resources available for how to use JMeter to write very complex performance tests where each of the operations could be specified exactly how many times to hit, and in what order, but for the purpose of this article the test case will be kept relatively simple and use this TradeScenarioServlet.

Using JMeter

To setup Jmeter and get it running to drive the performance test:

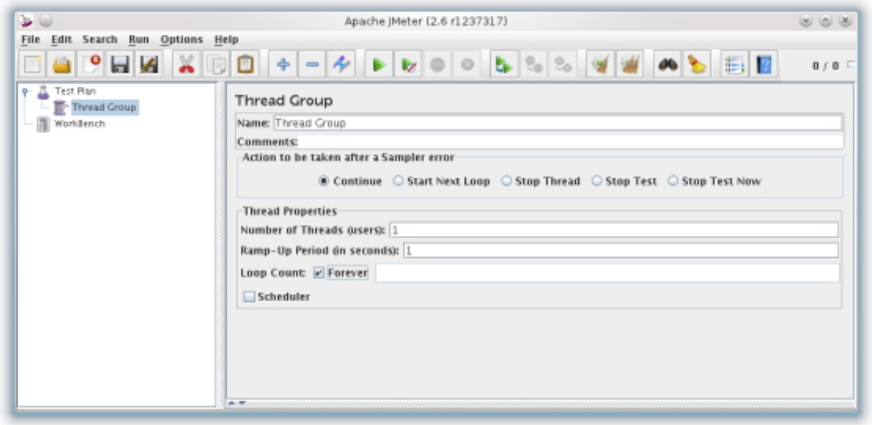
1. Install Jmeter. Pointing to your **java** directory, launch JMeter from the <JMeter_Home>/bin/ directory using the `jmeter.sh` or `jmeter.bat` script. You should see a panel similar to Figure 2.

Figure 2. JMeter default view



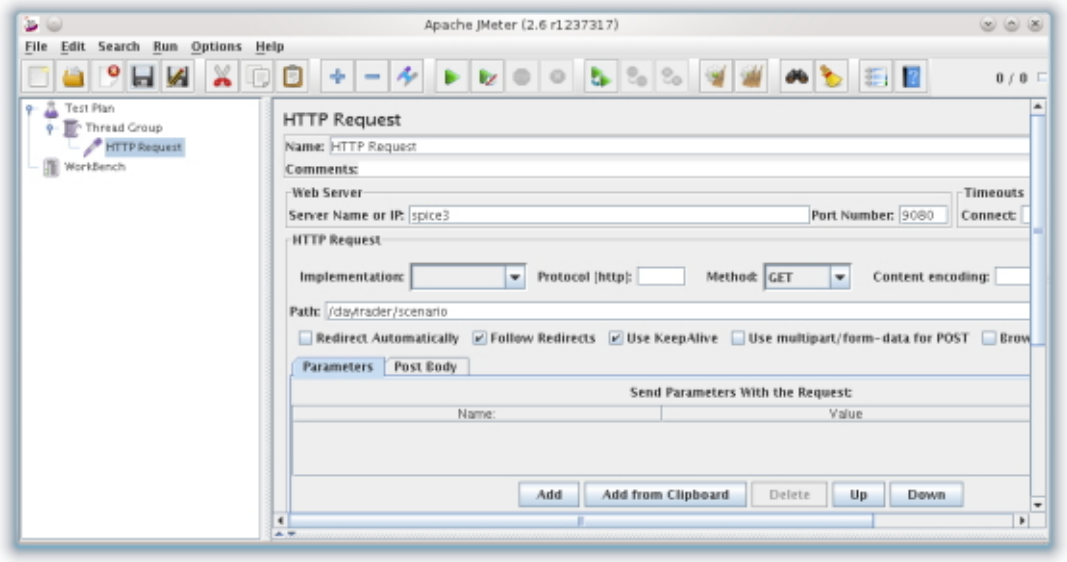
2. Right click on **Test Plan** and go to **Add > Thread Group**. This is where you define the number of users to drive load with. For starters, use these values (Figure 3):
    - Number of Threads (users): `1`
    - Loop Count: **Forever**

Figure 3. JMeter Thread Group view



3. Right-click on the thread group you just created, and go to **Add > Sampler**. A sampler defines the type of load you want to drive. There are samplers for HTTP requests, JMS requests, Web services messages, and so on. The JMeter user manual documents each of the available samplers. Because DayTrader supports web-based traffic, this example uses the **HTTP Request**. Fill out the values for the hostname, port, and path according to your environment (Figure 4).

Figure 4. HTTP request



4. Right click on the HTTP request you just created, and select **Add > Timer**. A timer adds “think time” in between requests. This simulates a user clicking on a page, and then pausing to read some information on the page, before making another request. There are many predefined timers you can choose from that range in complexity from a constant fixed timer to a gaussian or poisson distributed timer. The JMeter user manual documents each of the available timers. For this simple example, just use a **Constant Timer** of 5 ms.
5. Right click on the thread group and go to **Add > Listener > Summary Report**. This will show you the response times and throughput results while the test is running.
6. Save the settings to a file so you can load them again later.

Running the test

Always make sure the application works through a browser before starting the load driving tool. When ready, click the green arrow or click **Run > Start**. JMeter should now be driving one client to the server path you specified, waiting 5 ms in between each request. If you click on the **Summary Report** you can view the results as the test runs.

You should notice the throughput increasing over time; this is called the “warm-up” period. The JRE needs some warm-up time to load all the classes and let the JIT make some optimizations. The throughput will ultimately stabilize and reach a fixed number (give or take a few requests per second). Depending on how complex your test is, this warm-up period could be 30 seconds or 30 minutes.

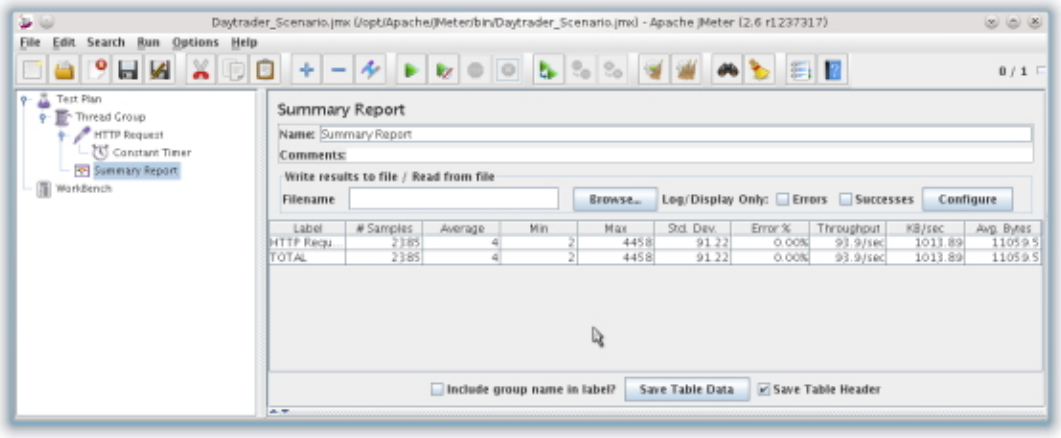
While the test is running, open a terminal (Linux or AIX) and run `vmstat 5` to display system metrics every five seconds (Listing 1).

Listing 1

```
[root@spice3 bin]# vmstat 5
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 8235164 104920 500956    0    0    13    11   44  154  8  5 86  0  0
 0  0      0 8235164 104928 500956    0    0     0     3 8030 4987  6  1 93  0  0
 1  0      0 8235040 104936 500956    0    0     0     3 7982 4944  5  1 94  0  0
 0  0      0 8233116 104936 500960    0    0     0     6 8126 5020  7  1 92  0  0
 0  0      0 8231068 104944 500960    0    0     0     6 7952 4939  6  1 93  0  0
 0  0      0 8231316 104952 500960    0    0     0     3 7761 4819  5  1 94  0  0
Example vmstat output showing ~7% CPU utilization.
```

If you're using Windows, right click on the task bar and select **Task Manager** and choose the **Performance** tab. Once the throughput in JMeter reaches a stable value, record the CPU utilization on the server running WebSphere Application Server and any other applicable servers, then stop the Jmeter test by clicking the red stop sign, or **Run > Stop**. The JMeter Summary Results view should look similar to Figure 5.

Figure 5. JMeter Summary Report view



In a spreadsheet, record the throughput result (93.9 req/sec) and the average response time result (4 ms). The Min, Max, and Std. Dev. response time results can also be beneficial to record if you want even more detailed information.

After recording all the results, select **Run > Clear** to clear the Summary Report results. This concludes the test for the single user. Now simply repeat the above steps by increasing the number of users to 2, then 4, then 8, and so on. You should observe the throughput increases each time as you add more users. Eventually, you'll observe the throughput stops increasing and may even start to decrease. Once that plateau (and possible degradation) is reached, the test can be stopped.

Analyzing the results

After completing the above steps, you should have a spreadsheet that looks something like Figure 6. (These particular results are very much dependent on the DayTrader application and the environment in which this test was run. Your actual results will likely look very different.)

Figure 6. Test Results – Table view

# of users	Throughput (req/sec)	response times (ms)	WAS CPU	DB CPU
1	10	2	1%	0%
2	24	2	1%	0%
4	44	2	1%	0%
8	86	2	2%	0%
16	162	3	3%	0%
32	315	4	9%	1%
64	624	6	12%	1%
125	1,208	10	23%	3%
250	2,366	13	45%	5%
500	4,613	23	82%	9%
1,000	5,559	32	99%	11%
1,500	5,587	34	99%	11%
2,000	5,565	34	99%	11%

Having the raw data in a tabular format like this is very beneficial, but it's also helpful to view the results in a graphical format. One of the best ways to visualize this is to use an XY scatter plot. Building a graph to chart the results makes it much easier to identify trends. Figure 7 charts the throughput and WebSphere Application Server CPU % versus the number of clients.

Figure 7. Test Results – Graph

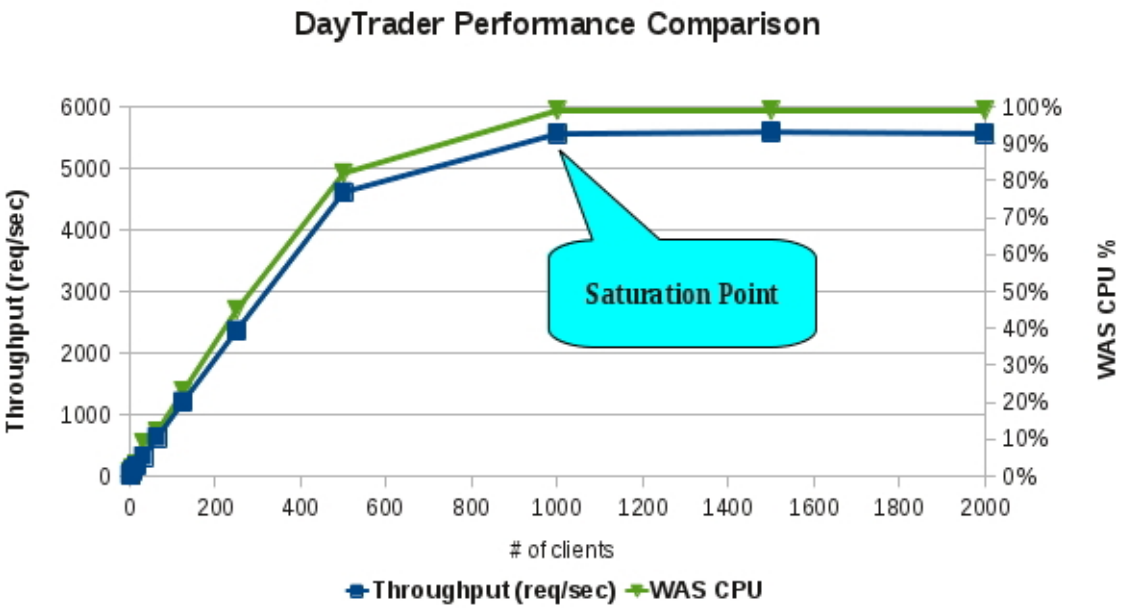


Figure 7 above shows some interesting characteristics. First, you see that the throughput curve and CPU % curve match closely together. Second, you see that the application throughput scales linearly from 1 client up to 500 clients. This is the desired result. However, somewhere in between 500 clients and 1,000 clients, the increase in throughput starts to slow down. (More tests could be done at this point with user loads in between 500 and 1,000 to find out exactly where this slowdown occurs.) Increasing the client load beyond 1,000 clients does not improve your overall application throughput. This is what's called the **saturation point**. This is a key value

that must be found during the performance test. The saturation point tells you that you've reached your maximum capacity for this application, tuning, configuration, and environment. Adding more users beyond this point will only increase client response times, but will not increase the overall application throughput. To achieve better performance beyond this point, an application code change, tuning change, or environmental change must occur.

DayTrader versus your application

DayTrader is not representative of all applications. Your application might reach the saturation point much sooner. If that is the case, it's likely indicative of an application bottleneck, which requires application analysis to remediate.

This type of testing and analysis is paramount in sizing and capacity planning discussions. It is only by identifying the saturation point that you can accurately estimate the total capacity needed to support a production environment. Too often, someone will say, “I need to support 10,000 users in this environment” and then run tests with that client load. Typically, this approach leads to a variety of overloaded conditions in one or more components, either in WebSphere Application Server or in other infrastructure components (network, database, and so on). A more productive approach is to determine what is achievable in terms of client load and throughput with a single application server and then proceed with run time and application tuning based on this. Once tuning is complete, you can then turn your attention to determining how many application server processes and physical servers are required to satisfy the scalability requirements.

Save a new test in JMeter with the # of Threads (users) you found as your saturation point. This can be used as a quick test to compare performance as you make changes to your application or environment, without going through the entire scalability test again. This isn't to say you should no longer execute the scalability test, but running with just the load at the saturation point is a great place to compare changes that likely won't show any difference at lower loads where the CPU is not fully utilized. A good practice is to run the saturation point load for minor application or tuning changes, and to repeat the entire scalability test for major application or environment changes.

Performance tools

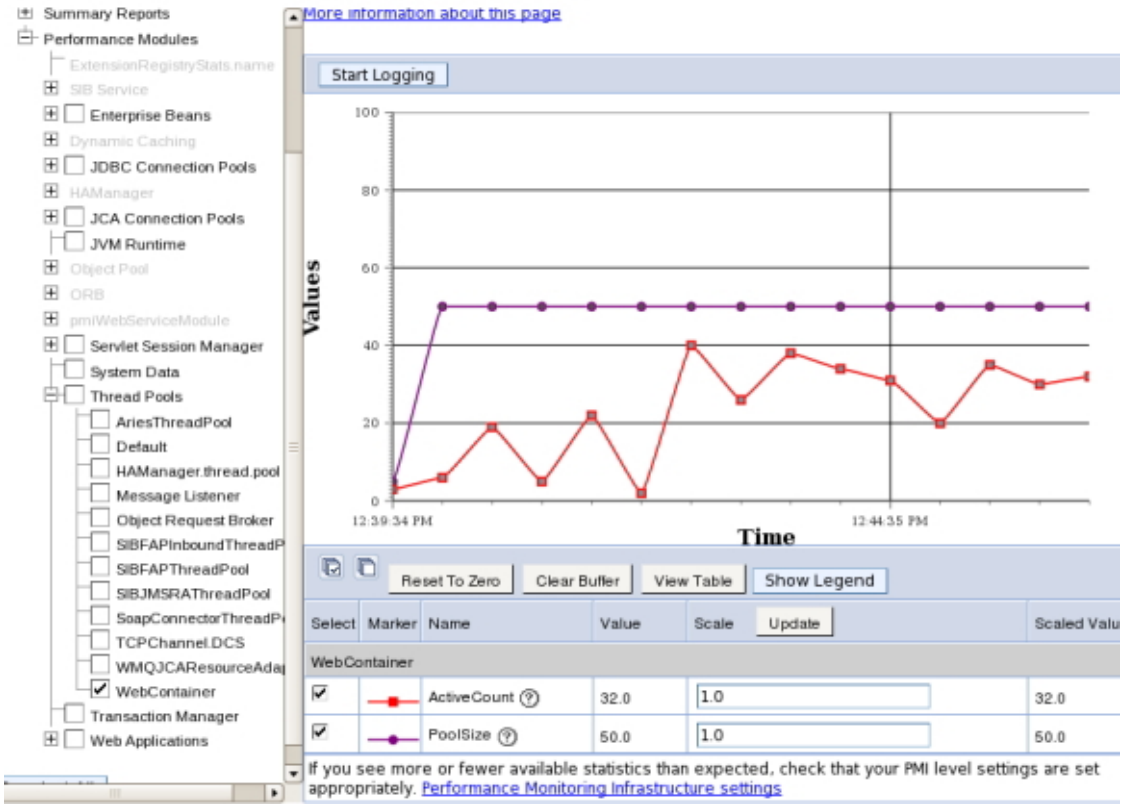
The sections above are prerequisites to doing any real analysis work. You must first understand how to generate repeatable performance results before looking for bottlenecks and performance improvements. Now that you're ready to start looking for improvements, here are two performance tools with which you should start your analysis:

- IBM Tivoli® Performance Viewer
  Tivoli Performance Viewer (in the admin console) is a very useful tool for monitoring WebSphere Application Server. This article really highlights the benefits of using Tivoli Performance Viewer to optimally tune an environment.

In the same manner, Tivoli Performance Viewer can be used to quickly check if there are any bottlenecks that could easily be removed by tuning.
Here are some simple steps to get started with Tivoli Performance Viewer:

1. Re-start the JMeter load with the number of users that were identified as the saturation point.
2. Login to the administrative console and select **Monitoring and Tuning > Performance Viewer > Current Activity**. Click on the server you want to monitor, and then expand **Performance Modules**. This will display a list of Performance Modules that are available to view.
3. Your application characteristics will determine which of these modules make the most sense to view. For the DayTrader example, or any other web-based traffic, start with the **Thread Pools > WebContainer** module. After checking that box, click the **View Module** button at the top. This will display a graph that looks something like Figure 8, populating more data automatically every 30 seconds as the JMeter test continues to run.

Figure 8. WebContainer PMI data

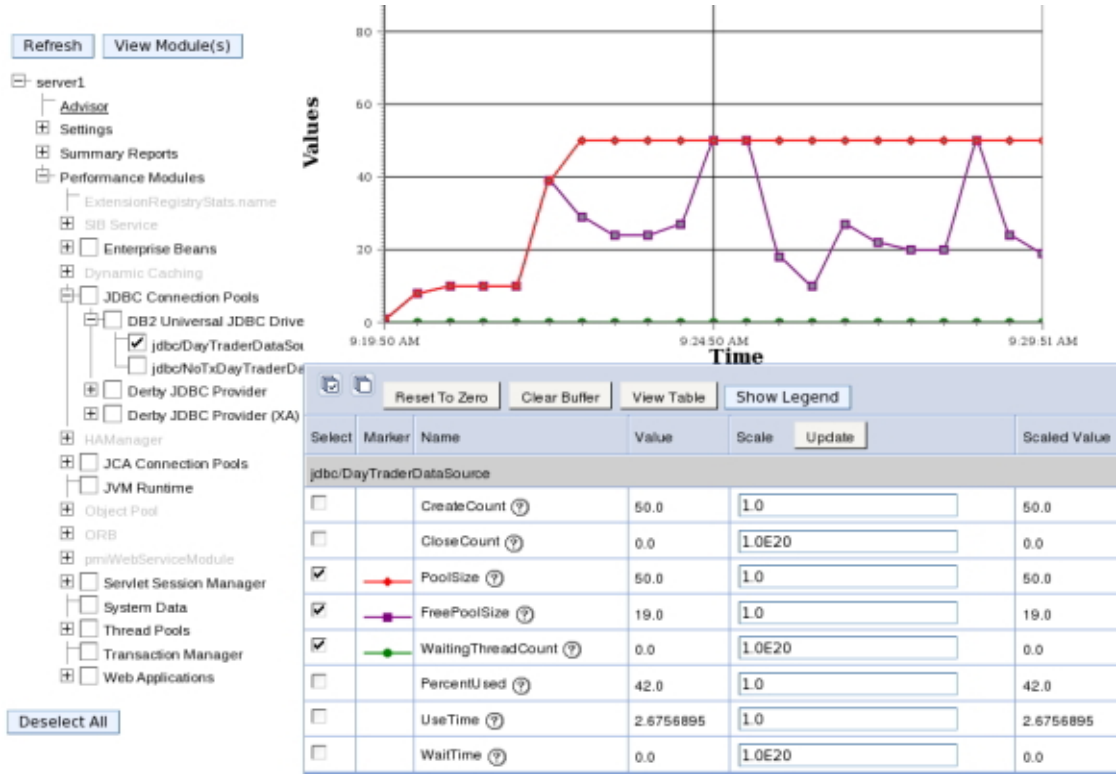


This example shows that the WebContainer thread pool size is at 50 threads, while approximately 32 are in use. This tells you that the WebContainer thread pool size is not a bottleneck at the current workload. If the active count was fluctuating between 45-50, then the WebContainer thread pool size could be a bottleneck. In that case, it would be best to increase the WebContainer thread pool size and repeat the test to see if

the performance improves. If the throughput does increase, you probably want to re-run the full scalabiltiy test again to re-establish your baseline.

4. Continue repeating step 3 with other modules that are applicable to your application. For a transactional application like DayTrader, another module you should view is the JDBC Connection Pool information. Expand **JDBC Connection Pools >** ***(your JDBC driver)*** and select your datasource JNDI name. Click the **View Module** button to display a graph that looks something like Figure 9.

Figure 9. DataSource PMI data



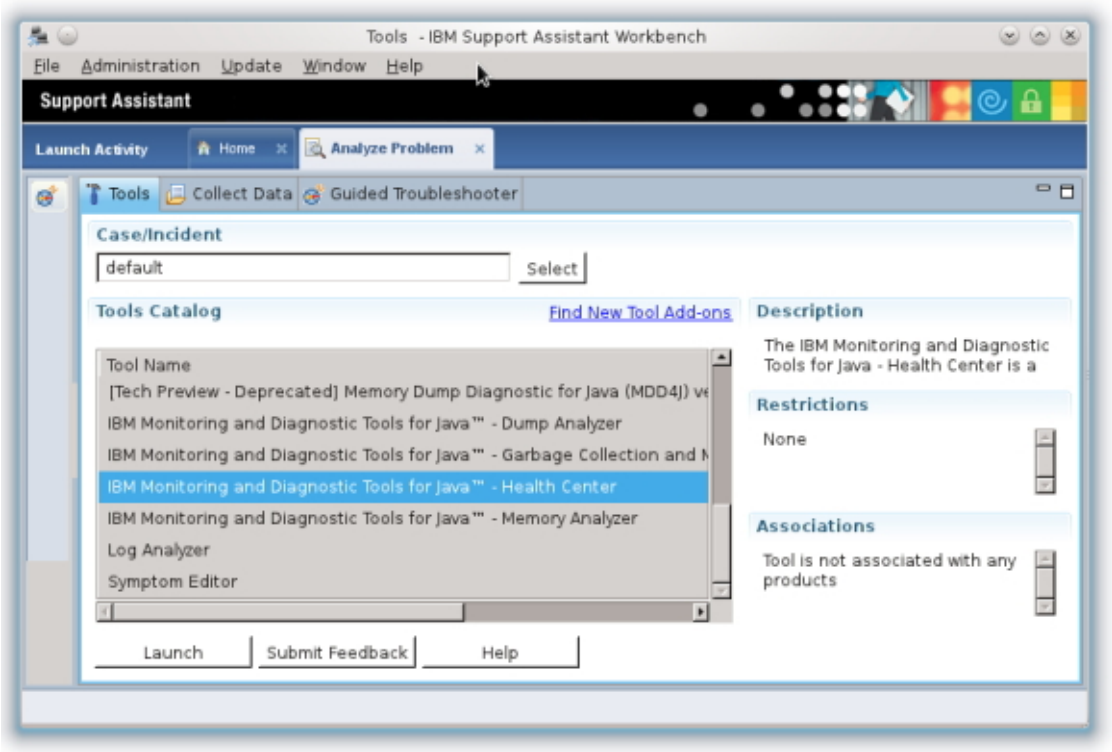
This chart shows a few different options charted rather than the default selections. The PoolSize, FreePoolSize, and WaitingThreadCount are all great metrics to review to ensure your connection pool isn't a source of contention and WebContainer threads aren't queued up waiting for a connection to the database. In the example above, the connection pool size is fixed at 50 connections (this is a tuning setting). The free pool size is fluctuating around 20, meaning that roughly 30 connections are active at a time. Together, this produces a Waiting Thread Count of 0, meaning that no WebContainer threads are waiting for a connection to the database. This verifies that the JDBC connection pool size is not a bottleneck. If the free pool size is 0 and the waiting thread count is greater than 0, then you might want to repeat the test with a higher connection pool size.

Continue this process with any other performance modules that are beneficial for monitoring your application. The WebSphere Contrarian: Preparing for failure has an extensive list of statistics that can be of great value to monitor. Once this exercise has been completed, you can move onto more detailed analysis with the IBM Health Center.

- IBM Monitoring and Diagnostic Tools for Java - Health Center
  The IBM Monitoring and Diagnostic Tools for Java - Health Center (hereafter referred to as Health Center, which is part of the IBM Support Assistant) tool is the recommend tool for detailed performance analysis on a WebSphere Application Server process. Health Center provides a wealth of knowledge about the performance of a server, including information about:
  - Memory usage
  - Garbage collection statistics
  - Method level profiling
  - Lock contention analysis.

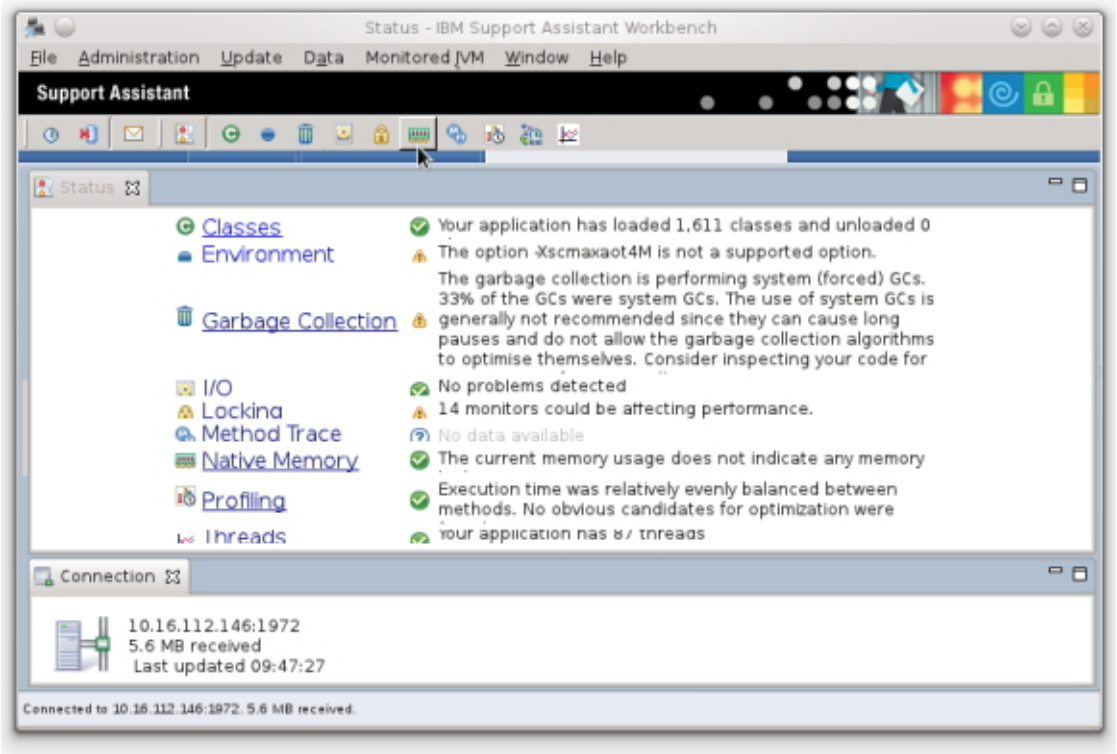
  Health Center is included as a tool in IBM Support Assistant (ISA), which is freely downloadable. Make sure you install ISA to a different machine than the application server machine, otherwise Health Center will take up resources away from the application server process, and your results might not be accurate. Health Center can be ran in an interactive mode, and in a “headless” mode where the information is saved to a file for later viewing. For this example, you'll use the interactive mode.
  To launch Health Center:
  1. Restart the WebSphere Application Server process you want to get detailed information on with the generic JVM argument `-Xhealthcenter`.
  2. Start ISA. When loaded, select **Analyze Problem**. (If you have not previously done so, you will need to tell ISA you are interested in tools for WebSphere Application Server.)
  3. Select **IBM Monitoring and Diagnostic Tools for Java – Health Center** and click **Launch** (Figure 10)

Figure 10. Launching Health Center from ISA



4. A connection wizard will display. Click **Next**. Specify the Hostame or IP address where the application server is running. By default, port 1972 will be used. If you have any security requirements, specify them here, otherwise click **Next**. If the hostname and port are found, click **Next** again, otherwise figure out why the connection didn't work. If successful, Health Center will look something like Figure 11.

Figure 11. Health Center default view



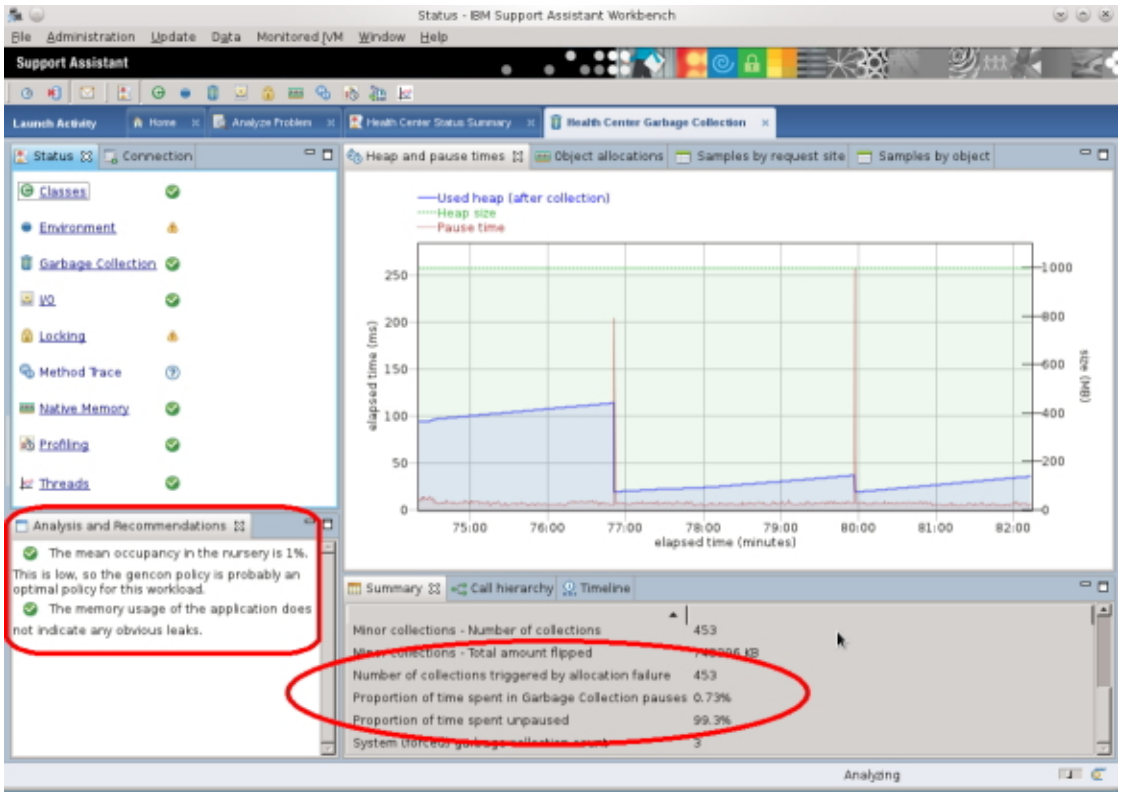
5. Maximize Health Center's screen and click around to get a feel for its features. At this point, you're ready to start some detailed performance analysis (which will be explored in the next sections. Go ahead and restart your JMeter load with the number of users found at your saturation point. Health Center will dynamically update as new information is available. Let the JMeter load run through your warm-up period before proceeding further.

Garbage collection analysis

The first step in any Java application performance analysis should always be studying the garbage collection statistics. With Health Center up and running, this is really easy to do.

Click on the **Garbage Collection** link in the Health Center window. A view similar to Figure 12 will display.

Figure 12. Health Center – Garbage Collection view



There are two key things to review first for entry level analysis:

- The **Analysis and Recommendations** section in the bottom left corner provides useful tips and information based on built in intelligence in Health Center. These tips can indicate garbage collection policy and heap size recommendations, observations about memory leaks or System.gc() calls, and more. In Figure 12, this section tells you that gencon is an optimal GC policy for DayTrader, and that the application does not appear to be leaking memory. That's always a good starting point.
- The **Summary** panel at the bottom of the window contains data for the most important statistics that you should be concerned with, starting with the “Proportion of time spent in Garbage Collection pauses.” This single statistic tells you what percentage of the time your application is stopped because garbage collection is occurring. This number should be as low as possible, ideally less than 2-3%. If this number is 10%, then you could see as much as 10% higher throughput by optimal tuning to your JVM heap sizes and garbage collection policy. As mentioned before, this case study is an excellent article to help guide you through that tuning process.

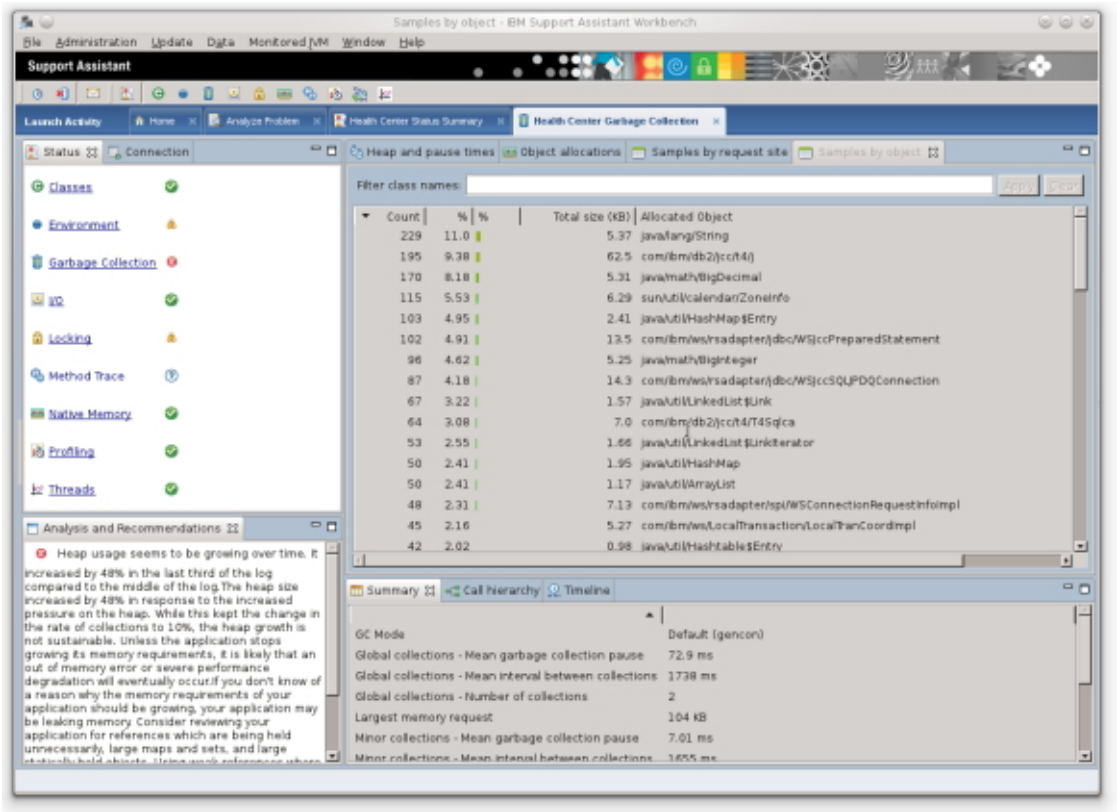
A few other tips to help you find the data you are most interested in:

- The X-axis in the chart in Figure 12 shows the elapsed time since server startup. You can change this to chart against the time of day. This can be useful

to correlate what activity was happening at certain times of the day. The X-axis can be changed by selecting the **context menu > Change Units > X-Axis > Time**.

- You can crop the data to eliminate the warm-up period to get a more accurate view of what's happening under the normal active conditions. To do so, select **Data > Crop Data** (to trim the start and finish) or just **Data > Reset Data** to clear any data up to this point.
- For more detailed analysis, click on the **Samples by object** tab. This enables you to see a breakdown of what objects are being allocated, how many are allocated of each type, and what the total size is. There's even an option to search by package or object name. An example is shown in Figure 13. Based on these results, it would probably be a good idea to review the application code to see if the usage of BigDecimal and BigInteger could be reduced.

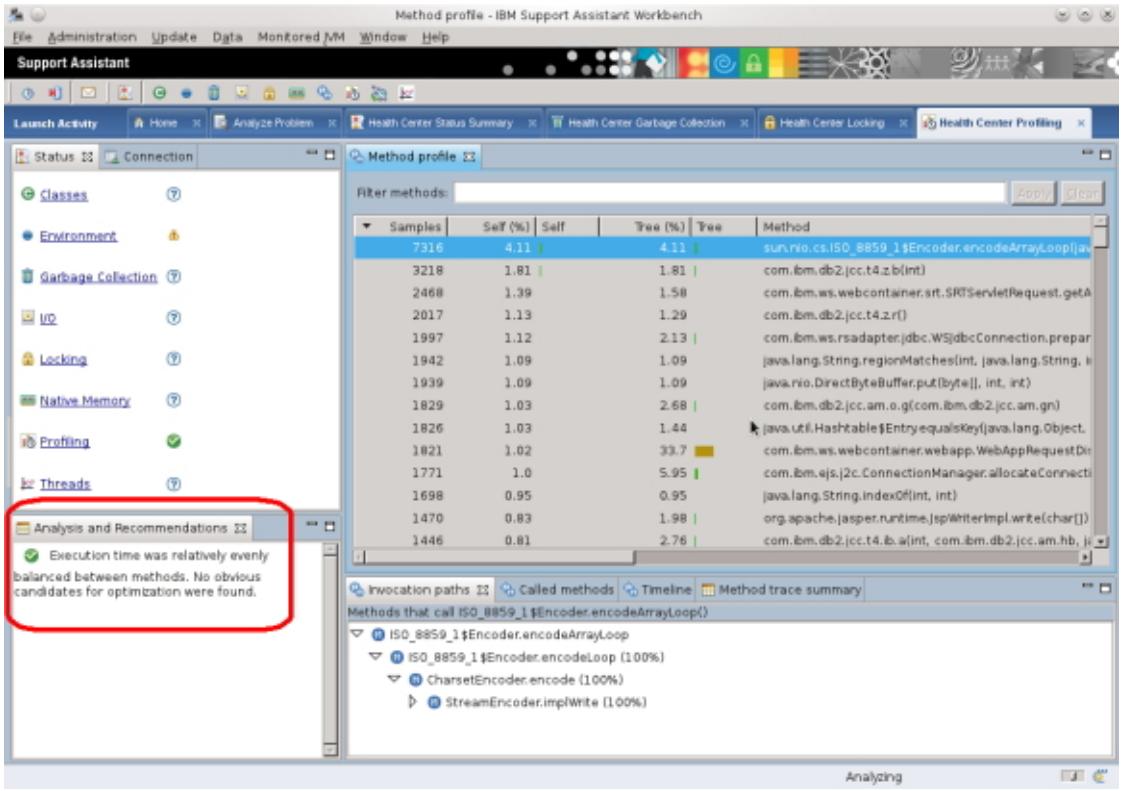
Figure 13. Health Center – Samples By Object view



Method profiling analysis

With the load driver still running, click on the **Profiling** link. This opens the Method Profiling view which looks like Figure 14.

Figure 14. Health Center – Method Profiling view



The Method Profile table shows which methods are using the most processing resources. This is a view of the entire JVM, not just your particular application, so this will include method information for database drivers, WebSphere Application Server containers, and so on. It's helpful to look at this view to get the larger picture of what's going on in the server. As with before in the garbage collection analysis section, one of the best places to start is the Analysis and Recommendations section. This panel will highlight any methods that were found to be consuming a much larger portion of the CPU cycles than the rest. In the example above, the tip says that there are no obvious methods for optimizing since all the cycles are pretty evenly split. If a method or two were pointed out here, the code for that method should be reviewed to see if any optimizations can be made, or the number of times it is called can be reduced.

To dig deeper, you need to have an understanding of how to interpret the data in the table. To assist with that, refer to the Health Center documentation by selecting **Help > Help Contents**, then scroll down and expand the **Tool: IBM Monitoring and Diagnostic Tools for Java - Health Center** book. The documentation states:

Methods with a higher Self (%) value are described as "hot," and are good candidates for optimization. Small improvements to the efficiency of these methods might have a large effect on performance. You can optimize methods by reducing the amount of work that they do or by reducing the number of times that they are called. Methods near the end of the table are poor candidates for optimization. Even large

improvements to their efficiency are unlikely to affect performance, because they do not use as much processing resource.

Here is a description of each of the columns in the table:

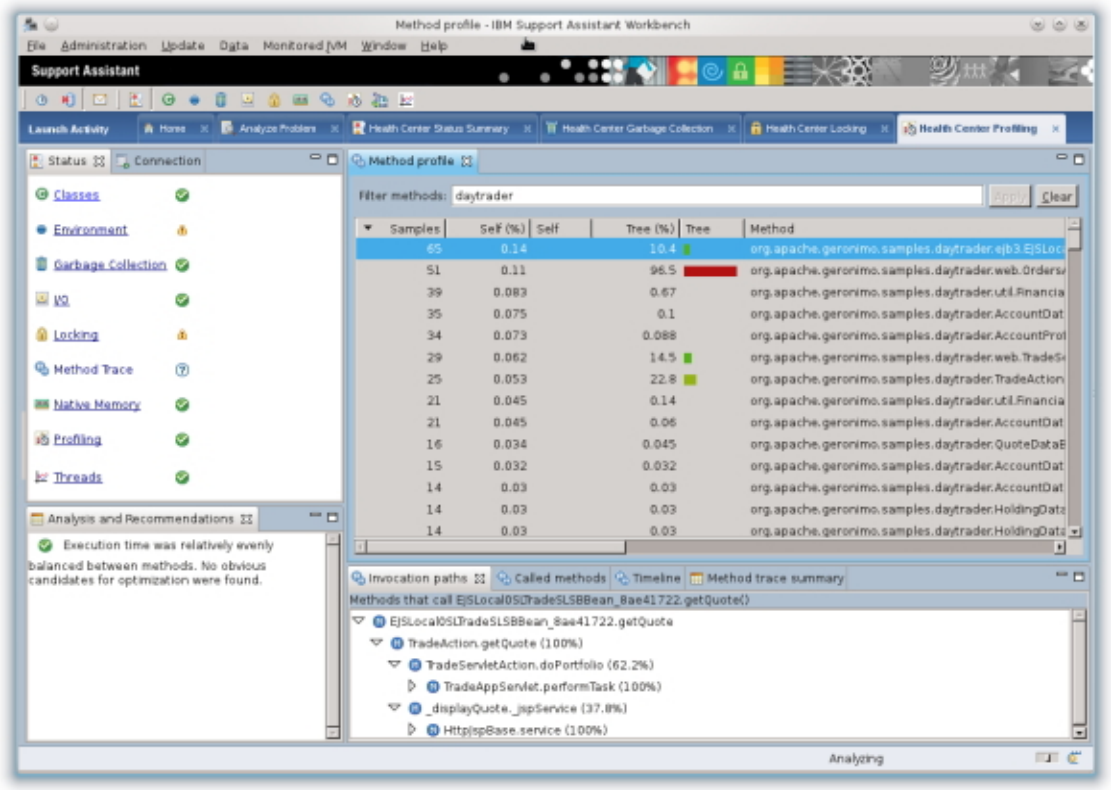
Table 1. Method profile table

Column	Description
Self (%)	The percentage of samples taken while a particular method was being run at the top of the stack. This value is a good indicator of how expensive a method is in terms of using processing resource.
Self	A graphical representation of the Self (%) column. Wider, redder bars indicate hotter methods.
Tree (%)	The percentage of samples taken while a particular method was anywhere in the call stack. This value shows the percentage of time that this method, and methods it called (descendants), were being processed. This value gives a good guide to the areas of your application where most processing time is spent.
Tree	A graphical representation of the Tree (%) column. Wider, redder bars indicate hotter method stacks.
Samples	The number of samples taken while a particular method was being run at the top of the stack.
Method	A fully qualified representation of the method, including package name, class name, method name, arguments, and return type.

Sort any of these columns by clicking the column header to sort in ascending or descending order. With this understanding, you can dig deep into the performance characteristics of your workload. Some useful tips for navigating through the table:

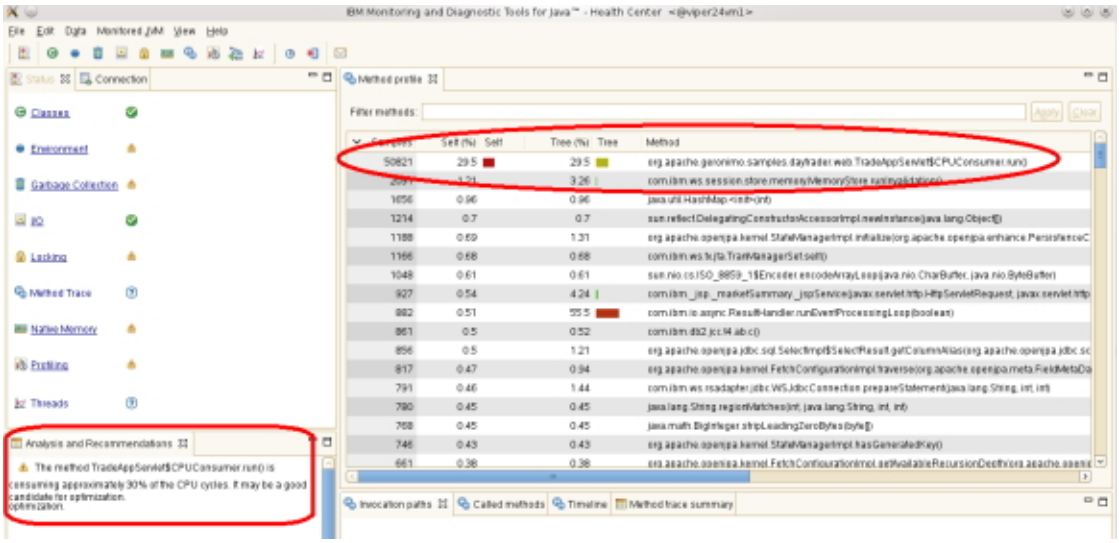
- Clicking on a row in the table will show you the full invocation path in the bottom panel of how the method got executed.
- The **Timeline** tab will show when the method was executed over the period for which the profiling has been active. This can be useful so you don't focus on something that was executed early on in the profile, perhaps during warm-up, but then goes away later on.
- The **Filter methods** text box is useful to search on specific classes and filter out the rest of profile. This should be used to drill down on details of your application only, to remove all the other non-application classes. As an example, Figure 15 shows the profile filtered on “daytrader” since all of the DayTrader application classes have “daytrader” in the package name. This enables you to focus on looking at the most resource intensive methods in just your application.

Figure 15. DayTrader filtered Method Profile view



If the application has a particular method that is consuming a lot of the CPU cycles, then Health Center will flag it is a good candidate for optimization in the Analysis and Recommendations panel. An example of that is shown in Figure 16.

Figure 16. Optimization candidate example



The Method Profiling view could be analyzed for days looking for performance improvements to an application. Since the output can be saved to a file, it makes comparing application changes extremely easy. Application developers could make a change that gets deployed and load tested with Health Center hooked in, and you

can compare the previous profile information to see if the methods changed by the developers have increased or decreased in processing requirements.

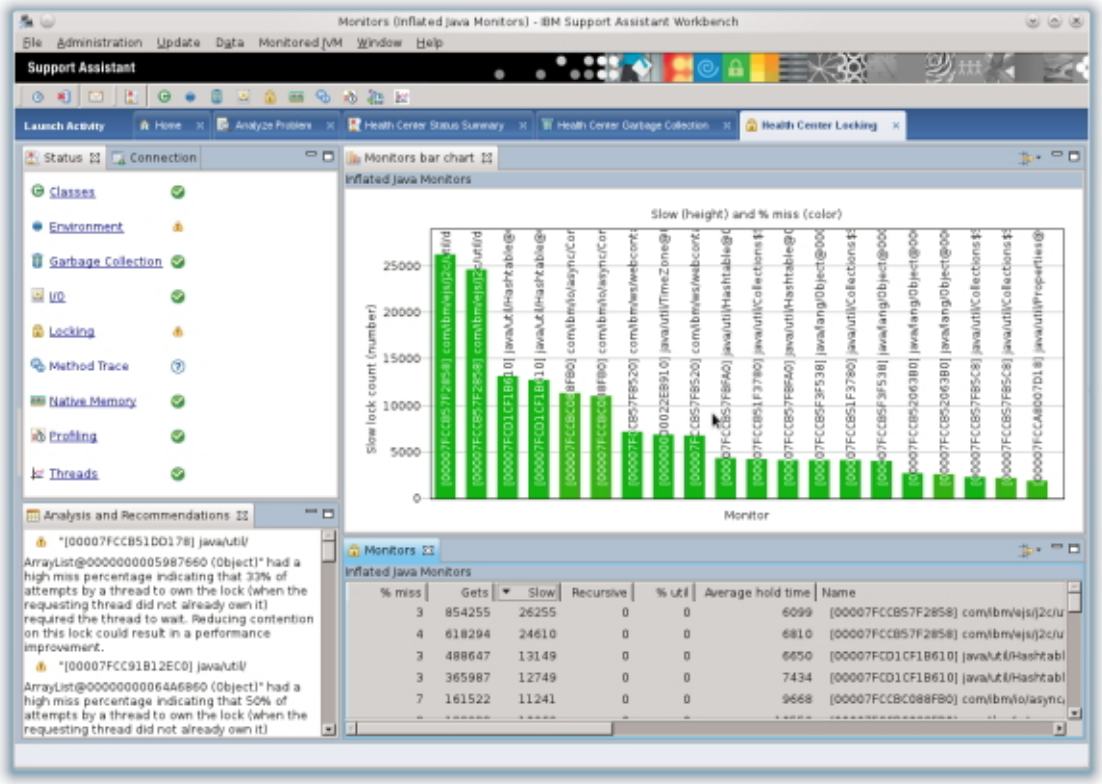
Locking analysis

Multi-threaded applications need to synchronize (or lock) shared resources to keep the state of the resource consistent. This consistency ensures that the status of one thread is not changed while another thread is reading it.

When locks are used in high-load applications that are deployed on systems with a large number of processors, the locking operation can prevent the application from using all the available processing resources. Imagine for a moment an application running on an 8-core machine with a major application code path being heavily synchronized such that only one thread can execute at a time. This could leave seven other threads waiting.

When running on large multi-core (4+) machines, lock analysis is essential to ensuring the application can scale up and utilize all the available hardware resources. (There's probably not much value in analysis here if the application is running on a single core machine.) The Locking perspective profiles lock usage and helps identify points of contention in the application or Java runtime that prevent the application from scaling. After clicking on the Locking link, a panel similar to Figure 17 should display.

Figure 17. Health Center – Locking view



At first glance, the Moniors view can be overwhelming to try and understand. However, the Health Center documentation describes this panel in great detail to help you understand these metrics. Table 2 describes the contents of the columns in the table

Table 2. Monitors

Column	Description
% miss	The percentage of the total Gets, or acquires, for which the thread trying to enter the lock on the synchronized code had to block until it could take the lock.
Gets:	The total number of times the lock has been taken while it was inflated.
Slow:	The total number of non-recursive lock acquires for which the requesting thread had to wait for the lock because it was already owned by another thread.
Recursive:	The total number of recursive acquires. A recursive acquire occurs when the requesting thread already owns the monitor.
% util:	The amount of time the lock was held, divided by the amount of time the output was taken over.
Average hold time:	The average amount of time the lock was held, or owned, by a thread. For example, the amount of time spent in the synchronized block, measured in processor clock ticks.
Name:	The monitor name. This column is blank if the name is not known.

The table lists every Java monitor that was ever inflated. The % miss column is of initial interest. A high % miss shows that frequent contention occurs on the synchronized resource protected by the lock. This contention might be preventing the Java application from scaling further.

If a lock has a high % miss value, look at the average hold time and % util. Some tips:

- If % util and average hold time are both high, you might need to reduce the amount of work done while the lock is held.
- If % util is high but the average hold time is low, you might need to make the resource protected by the lock more granular to separate the lock into multiple locks.

Conclusion

Getting started in performance testing and analysis can be difficult at first without the right tools and knowledge. However, this article showed that there are some very simple steps that can be followed to ensure proper performance testing and that application bottlenecks have been removed such that it's performing as efficiently as possible.

Even though DayTrader might not resemble your application, the methodologies described in this paper for performance testing and identifying bottlenecks are the

same. Testing with small user loads for slow periods up to high user loads for peak usage periods is vital to understanding your application's characteristics. Recording key metrics for comparisons as application or environment changes are made is essential to understanding where performance degradations might be coming from. Finally, the IBM Health Center tool makes performance analysis a breeze with garbage collection, method profiling, and lock profiling views to help you ensure the application is performing as efficiently as possible.

Resources

- WebSphere Application Server Performance
- IBM Monitoring and Diagnostic Tools for Java - Health Center Version 2.0
- Tutorial: Hello World: Rational Performance Tester Tutorial
- Book: Performance Analysis for Java Websites
- How well does traditional performance testing apply to SOA solutions?
- The WebSphere Contrarian: Preparing for failure
- Case study: Tuning WebSphere Application Server V7 and V8 for performance
- The WebSphere Contrarian: Less might be more when tuning WebSphere Application Server
- IBM developerWorks WebSphere

About the author

David Hare

David Hare an Advisory Software Engineer with the WebSphere Application Server Performance and Benchmarking organization in Research Triangle Park, North Carolina. His primary focus has been on the DayTrader performance benchmark, performance tuning, and the brand new Liberty Profile.

Key features and capabilities of IBM Worklight to accelerate your mobile development

Harish Shenoy, IT Architect, IBM

Summary: IBM® Worklight provides a complete platform for developing, deploying, hosting, and managing mobile enterprise applications. Worklight addresses all the requirements for mobile application developments and provides tools and efficiencies to help at every stage of the development and deployment process. This article offers an overview of IBM Worklight V5 with a checklist of some of the new and key features that will help you launch and achieve your mobile development objectives. This content is part of the IBM WebSphere Developer Technical Journal.

Date: 01 Aug 2012
Level: Intermediate
PDF: A4 and Letter (968 KB | 12 pages) Get Adobe® Reader®

Introduction

IBM Worklight V5 provides an open, comprehensive, advanced platform for developing mobile enterprise applications for smartphones and tablets. As a complete, open standards-based platform that leverages technologies such as HTML5, CSS3, and Apache Cordova, Worklight helps organizations of all sizes efficiently develop, connect, run, and manage HTML5, hybrid, and native mobile applications.

Worklight supports all stages of the mobile development lifecycle, including application building, deployment, execution, and management, and provides tools to help at each step along the way.

Leveraging standards-based technologies and tools, Worklight ships with a comprehensive development environment, mobile-optimized middleware, and an integrated management and analytics console, supported by a variety of security mechanisms. Worklight enables the creation of rich, cross-platform applications without the use of code translation, proprietary interpreters, or unpopular scripting languages. It also reduces time to market, development cost, and overall complexity, enabling you to provide a better user experience across a broad array of mobile devices.

IBM Worklight is part of the IBM Mobile Foundation family of products, which provides the essential elements needed for complete mobile development, deployment, and management within a business.

With Worklight, you can efficiently manage the development and maintenance of multiple, optimized versions of your mobile applications. Mobile apps targeted to different mobile operating systems (such as Android, iOS, and others) can be built in a uniform fashion. Because a large percentage of the common code base is shared across different operating systems, the ability to rely on ready-made components and to share device-independent sections of code drastically reduces development time. Mobile applications delivered through Worklight leverage common web technologies across multiple devices, without sacrificing the ability to tap the power of native technologies and tools, such as Objective-C, xCode, or Android. As such, native code can also be added to applications targeted to specific operating systems, if desired. Features of specific mobile devices can be accessed by your applications using Apache Cordova or native APIs combined with the shared common code base.

This articles presents a high level checklist of some of the latest key features of Worklight to help you become familiar with its capabilities and understand how it can help you launch and support fast and effective mobile application development.

Back to top

An overview of Worklight

The Worklight platform consists of four main components:

- **IBM Worklight Studio** is an Eclipse-based integrated development environment (IDE) that enables you to perform all the coding and integration tasks required to develop a fully operational mobile application for various mobile operating systems. Eclipse users will find Worklight Studio easy to use it with little or no additional learning required for develop-ing mobile applications with the assisted code development features.
- **IBM Worklight Server** is a Java™-based server that is a scalable gateway between applications, external services, and the enterprise back end infrastructure. The server contains security features to enable connectivity, multi-source data extraction and manipulation, authentication, direct update of web and hybrid applications, analytics, and operational management functions. Worklight Server supports IBM WebSphere Application Server and Apache Tomcat run time environments for executing Worklight applications.

IBM Worklight Device Runtime Components consist of client-side run time code that embeds server functionality within the target environment of deployed applications.

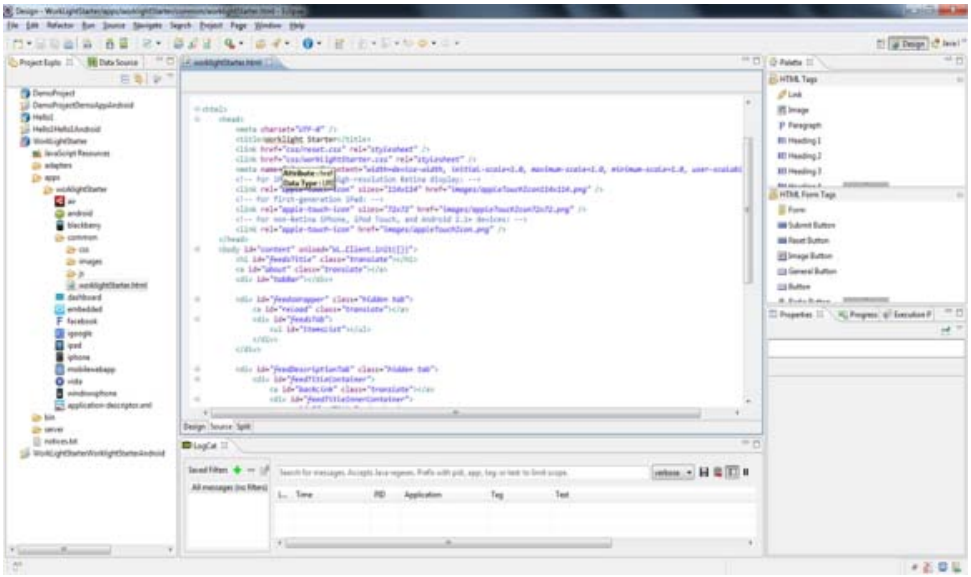
- **IBM Worklight Console** is a web-based administrative console that supports the ongoing monitoring and administration of the Worklight Server and its deployed applications, adapters, and push notifications. You can also use the console to manage different versions of mobile apps and send any notifications to application users.

Through these components, Worklight provides a broad array of features and capabilities. Among these are:

- **Single assisted development environment for building cross platform applications**

  Worklight Studio provides an environment for developing mobile applications for all the supported mobile platforms. Applications can be developed as mobile web applications, or for specific operating systems including Android, iOS, Black-berry, and Windows. Figure 1 shows a view in the Worklight Studio development environment.

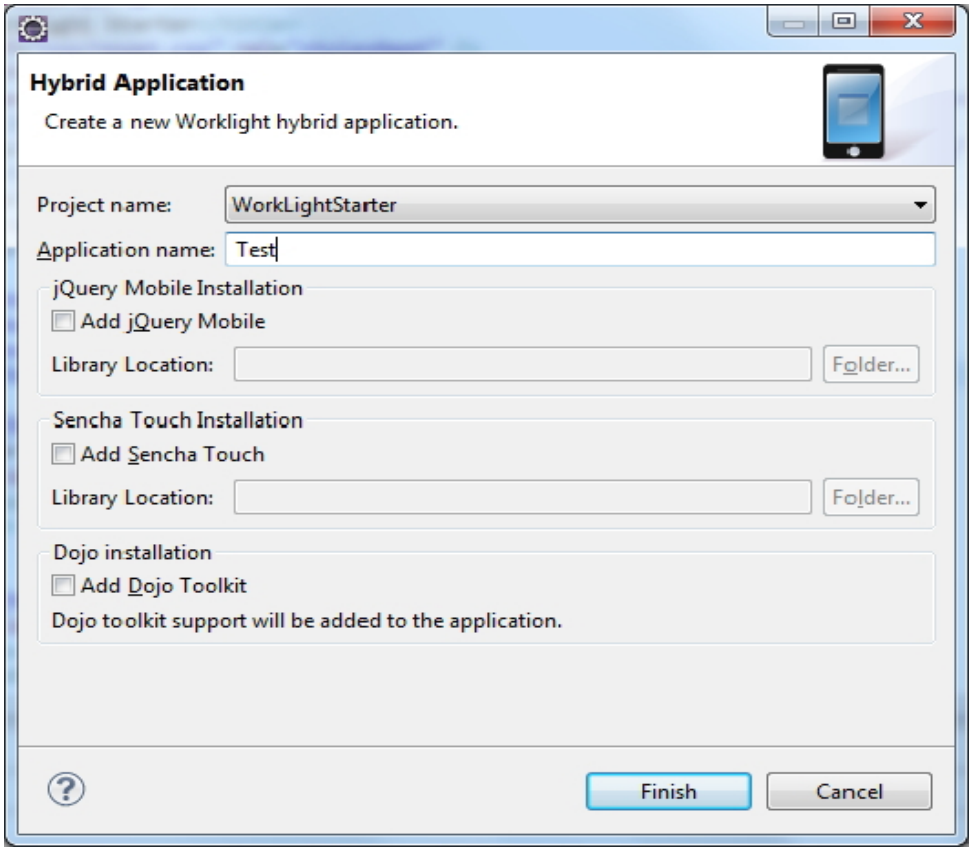
Figure 1. Worklight Studio IDE



- **Open approach to third-party integration**

  Third-party JavaScript™ libraries, such as JQuery Mobile, Sencha Touch, or Dojo Toolkit, can be integrated seamlessly with the Worklight Studio development environment. This enables you to reuse existing assets built using these libraries. Worklight ships with the Dojo library that can be used in any application. To use JQuery and Sencha libraries, they must be configured during the creation of your mobile application. Being able to use existing assets in the development of new mobile apps can potentially shorten the development cycle for mobile applications. Figure 2 shows a Worklight application creation wizard that provides options for integrating libraries.

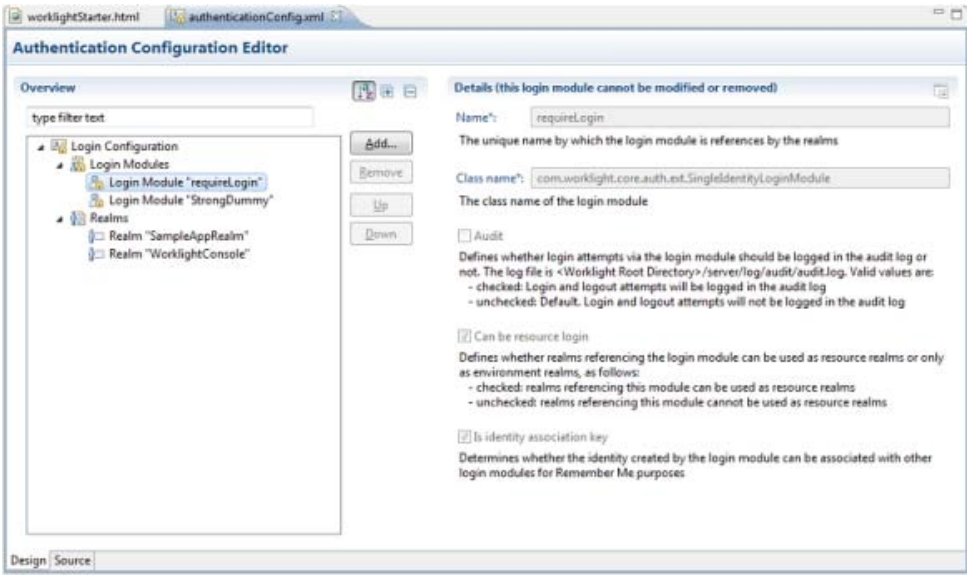
Figure 2. Application creation wizard



- **Strong authentication framework**

  Worklight has a built-in authentication framework that you can configure and use with very little effort. Authentication can be form-based, cookie-based, HTTP header-based, or adapter-based. Worklight Studio provides an editor for you to view and edit the authentication configuration for any application. Worklight also provides the option for you to create a custom authentication framework. Figure 3 shows the Authentication Configuration Editor in Worklight Studio.

Figure 3. Authentication Configuration Editor

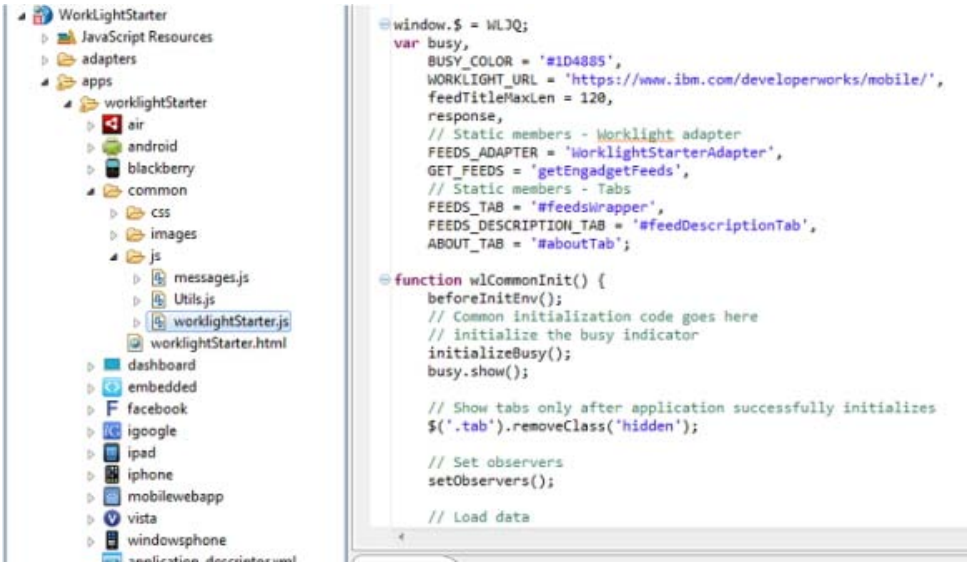


- **Share common code base across operating systems**

  A common code base shared across all supported mobile operating systems is a key capability of Worklight, When you create an application in Worklight, it cre-ates a default application code base location with a folder called common. Most of the application code required for common cross-platform functionality can be coded and tested using this common code base. Once the common functionality is completed, the platform-specific requirements for the application can then be added under the platform-specific code base. This helps you achieve maximum reuse of common code, and avoid redundant coding when developing applications that are even targeted for multiple operating systems. Worklight makes

managing and supporting code easy and convenient. Figure 4 shows the Worklight applica-tion file structure in which the application code in the common folder can be shared across applications for all platform environments.

Figure 4. Worklight application structure

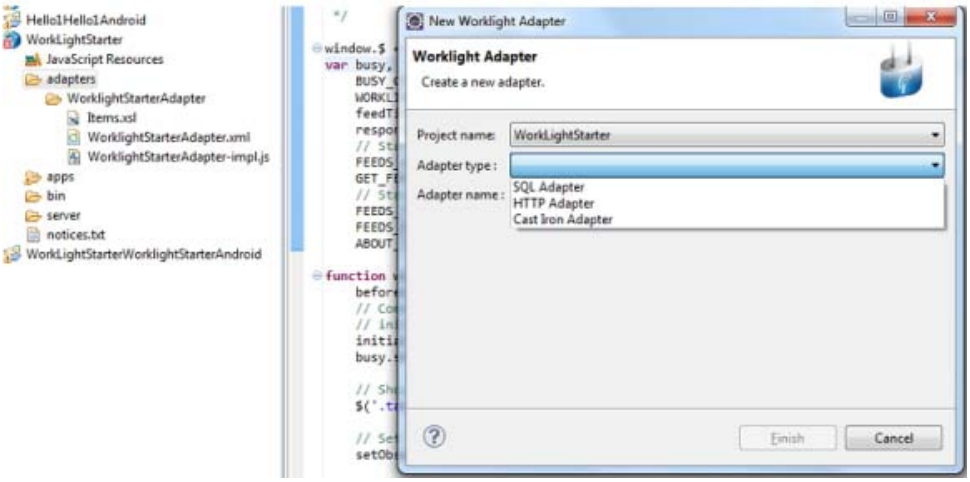


- **Enterprise back end connectivity**

  Worklight adapters enable the integration of back end systems with applications developed with Worklight. Out of the box adapters are available for connecting to enterprise back end systems using database, web services, or Cast Iron. Adapters can also be created easily in Worklight Studio. Three types of adapters can be created:
  - SQL adapter
  - HTTP adapter
  - Cast Iron adapter.

  Worklight Studio also offers the option of testing adapter functionality before they are used by client applications. Figure 5 shows the adapter creation wizard in Worklight Studio, which enables you to easily connect mobile applications to en-terprise back ends and reuse existing services.

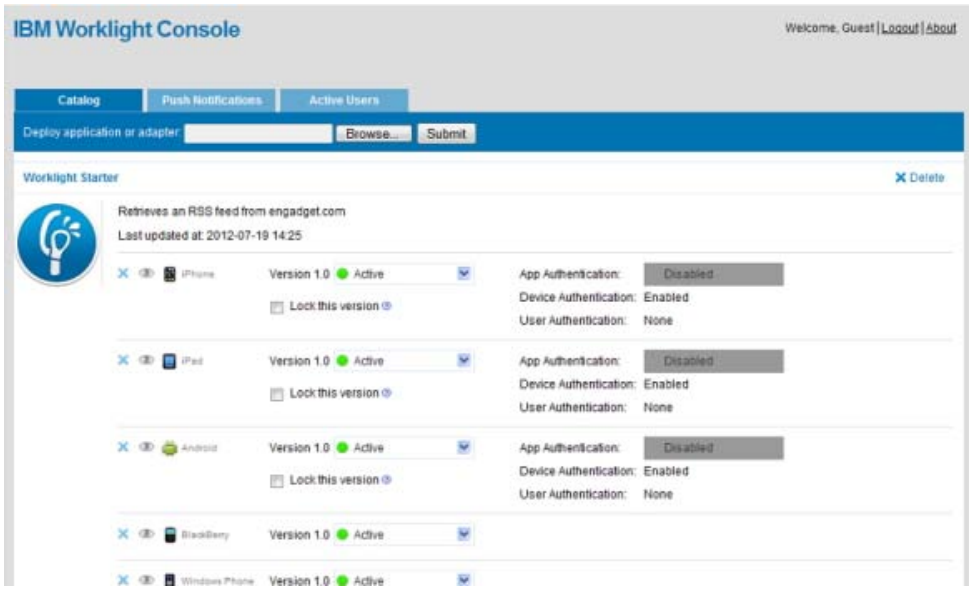
Figure 5. Adapter creation wizard



- **Administration console to manage all applications**

  Worklight provides a browser-based management console that you can use to de-ploy, manage, and otherwise administer all applications and adapters through a single interface. The admin console is used for managing applications for all the supported mobile operating systems. You can also manage multiple application versions with the admin console, along with push notification configurations and active user reports. Figure 6 shows the Worklight console application manage-ment view.

Figure 6. Worklight Console



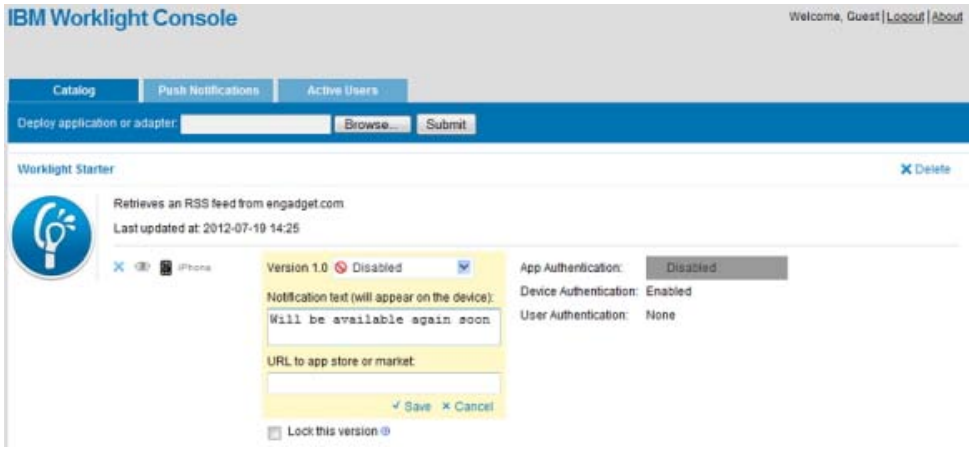
- **Unified push notifications**

  Push notification is a mechanism that can send notifications to devices where Worklight applications are installed. Push notifications can be configured to send notifications to devices regardless of whether applications are running on devices in the foreground or not. Configurations can be setup to send the notifications to devices using Android C2DM or Apple APNS for devices using Android or iOS, respectively. Worklight enables you to send notifications to all devices, a subset of devices, or to a single device.
- **Encrypted offline availability**

  Worklight provides the ability to encrypt data that needs to be stored as such on the device. Worklight's API framework detects application connectivity status so that the application can take appropriate action if offline. Worklight's encrypted cache mechanism can be then used to store sensitive data in encrypted format on devices. This enables you to address data security requirements for information that needs to be stored on devices. Offline application availability and offline au-thentication is possible with these embedded features. This feature addresses requirements for building secure mobile apps with consideration for network availability.
- **Direct updates and remote disablement**

  Once Worklight applications are installed on devices, application updates can be applied directly if there is a change in an application that is deployed on Work-light Server. This feature is known as direct update. When a Worklight applica-tion starts on foreground on the device, it checks for updates from Worklight Server. If updates are available, Worklight Server pushes the application updates to the device so that it is running the latest version of the application. This feature saves much time and effort usually spent releasing updated versions to applica-tions to devices. The Worklight Console also provides an option to disable any application in case usage of that application needs to suspended for some reason. When disabling, notifications related to availability can also be pushed to applica-tion users to keep them aware of the current availability status. Figure 7 shows how an application can be disabled remotely, along with a notification message and download link being sent to users. This features addresses requirements for sending modified features or fixes to applications.

Figure 7. Application management view of Worklight Console

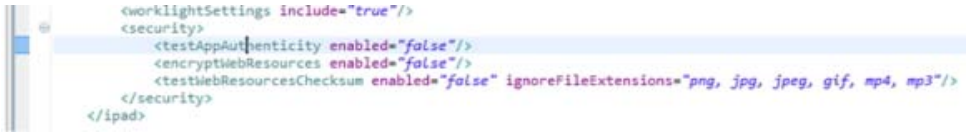


- **Application security**

  Worklight provides multiple features supporting application security. Authenticity of any application connecting to Worklight

Server can be configured so that ap-plications that are modified post distribution are prevented access from Worklight Server. This feature can be configured using the testAppAuthenticity property in the application descriptor configuration of any application. This feature secures the Worklight application in case it is redistributed with changes that might not be authentic. Code obfuscation is possible to prevent the redistribution of applica-tions with unauthorized modifications. Figure 8 shows a snapshot of a Worklight application configuration file with a security configuration option.

Figure 8. Worklight configuration file snapshot for application security



- **Data collection for analytics**

  Worklight provides features to collect data for analytics. Data can be collected at both the server level and device level. The data collected can be configured to get the analytics data for various requirements using different reporting tools. Work-light's reporting capability can help you find active users from the admin console. Basic analytics reports, such as the application-specific reports based on new downloads, application access patterns, daily visits, and daily hits can be config-ured using the Business Intelligence Reporting Tool (BIRT) Eclipse plug-ins. The analytics data can then be exported to enterprise report systems. This feature addresses enterprise wide analysis and reporting requirements. Figure 9 shows the Active User Report view in Worklight Console depicting the users logged in for last 30 days.

Figure 9. Worklight Console Active User Report

Back to top

Conclusion

This article provided a high level checklist of some of the key features and capabilities of IBM Worklight. Use the Resources below to decide where you'd like to learn more about Worklight so you can begin to apply the advantages Worklight provides to enhance your competitive edge in mobile enterprise applications.

Resources

Learn

- IBM Worklight user documentation
- IBM Worklight features and benefits
- IBM developerWorks Mobile zone
- IBM developerWorks WebSphere

Get products and technologies

- Download IBM Worklight Developer Edition
- Evaluate IBM Worklight Developer Edition 5.0

Discuss

- IBM Worklight Forums

About the author

Harish Shenoy is an IBM Certified SOA Solution Designer. He is currently working in BPTSE team IBM India Software Lab providing technical solution architecture in Application Connectivity and Application Infrastructure. His expertise includes IBM Worklight, WebSphere Application Server, WebSphere eXtreme Scale, WebSphere Process Server, WebSphere Message Broker and BPM. He is also certified in WebSphere Process Server, WMQ,WebSphere Message Broker and WebSphere Application Server.

Rate this article

Comments

Back to top

Print this page Share this page Follow developerWorks

About Feeds and apps Report abuse Faculty
Help Newsletters Terms of use Students
Contact us IBM privacy Business Partners
Submit content IBM accessibility

The Support Authority: An overview of the IBM Support Assistant 5.0 Beta

Jim McVea (jimmcvea@us.ibm.com)
Technical Architect, IBM Support Assistant
IBM

Paul Blizniak (The_Bliz_@us.ibm.com)
Software Engineer
IBM

Skill Level: Intermediate

Date: 01 Aug 2012

This article provides an introduction to the Beta 1 release of IBM® Support Assistant 5.0 and offers an overview of the installation, configuration, and new key features and concepts. Users in a Support Analyst role or who are otherwise responsible for troubleshooting and diagnosing software problems will benefit from the new IBM Support Assistant strategy and tooling available.

View more content in this series

In each column, The Support Authority discusses resources, tools, and other elements of IBM® Technical Support that are available for WebSphere® products, plus techniques and new ideas that can further enhance your IBM support experience.

Enhancing your support experience

IBM Support Assistant is a free application that provides features for self-help problem determination and a platform for obtaining diagnostic tools. These features and tools streamline troubleshooting by making it easy to organize, analyze, diagnose, and eventually solve problems that might occur with IBM software or deployed applications.

The new Beat release of IBM Support Assistant 5.0 features many usability improvements over the current version 4.0 client application, and in addition has adopted a cloud-based approach. As a server application with a rich, browser-based front end, you can offload the analysis of data to a server class machine and avoid some of the system resource limitations that might be experienced with a typical laptop or desktop system.

A server-based install also enables an administrator to install a single instance of IBM Support Assistant that can then be used by a group of users and accessed very simply by pointing a browser to it where resources, files, information, and tools can be shared.

The intent of this article is to introduce the IBM Support Assistant 5.0 strategy and provide a basic understanding of the tool, its key concepts, and core functions.

Getting started

System requirements

Before installing and using IBM Support Assistant 5.0, you must ensure that the target system's hardware, operating system, and browsers that will access IBM Support Assistant meet the minimum requirements.

Installation

There are two installation options:

- **EAR deployment into an existing WebSphere Application Server**
  If you have an existing WebSphere Application Server, you can deploy the IBM Support Assistant 5.0 EAR and associated tool WAR files into it. This option is the most flexible, because you have more control over how the application and its tools are installed.
  The IBM Support Assistant 5.0 EAR is supported on WebSphere Application Server V7.0.0.21 or higher. Using the WebSphere Application Server admin console, you can install the ISA5.ear file just as you would install any other enterprise applications. The fast path install via the administrative console can typically be used. You will also need to install the tool WAR files from the download package as separate web applications. (The server-based tools use the WebSphere Application Server JRE to execute in their own JVM process.) Quick steps for installing IBM Support Assistant as an enterprise application, including system requirements, are available.
- **All-in-one solution**
  This solution has everything you need to run the IBM Support Assistant application, including an embedded application server and Java™ runtime. This option is the easiest to use because it models the "unzip and go" scenario.
  How you install and start the application depends on which operating system platform you are targeting. (Be aware that IBM Support Assistant 5.0 does not support installation into a directory path that contains '.' characters or space characters.) The server-based tools included in the package use the IBM Support Assistant JRE to execute in their own JVM process. Quick steps for installing IBM Support Assistant using an embedded server, including system requirements, are available.

Either option can be installed onto your own desktop machine, as well as on a shared server. Installing onto your desktop is a good option for trying out IBM Support

Assistant before deploying out to a team environment, or if you will be using the application as a single-user instance.

Optional configuration

Several elements of the IBM Support Assistant 5.0 runtime can be changed from their installation default values. These include location of case files, JVM max heapsize, and port numbers. The ReleaseNotes.txt file, found at the root directory of any of the download archive files, includes this configuration information, along with Quick Start information, known restrictions, and requirements).

Using IBM Support Assistant 5.0

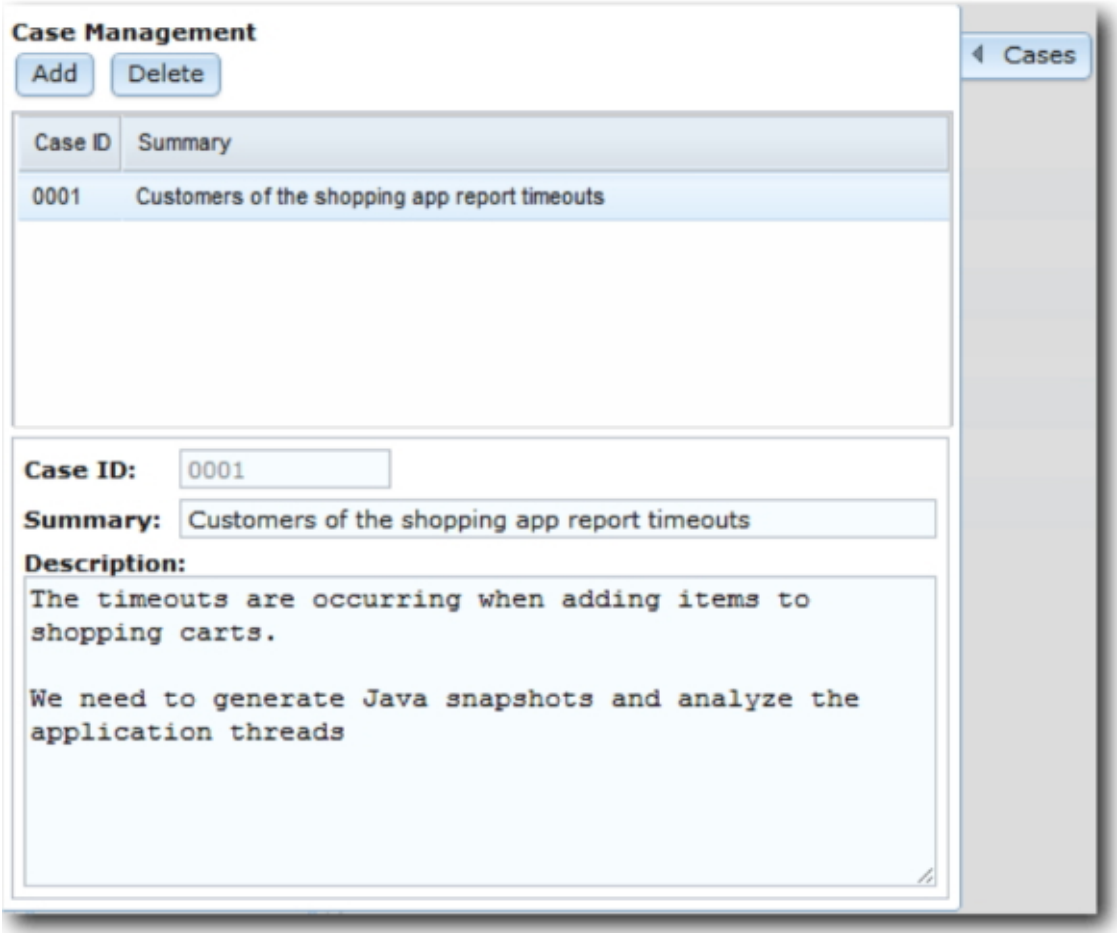
Once you have the application installed and any optional configuration completed, you're ready to get started using the tool. This is easy. Assuming you're using default values, open a browser to `http://<hostname>:10911/isa5` and try it out!

Key features

IBM Support Assistant 5.0 emphasizes problem determination through diagnostic tooling in a multi-user environment. There are several new key features and concepts that are surfaced in the application to support this focus. Many of these features will be highlighted as the tool's components are described through the typical usage scenarios detailed below.

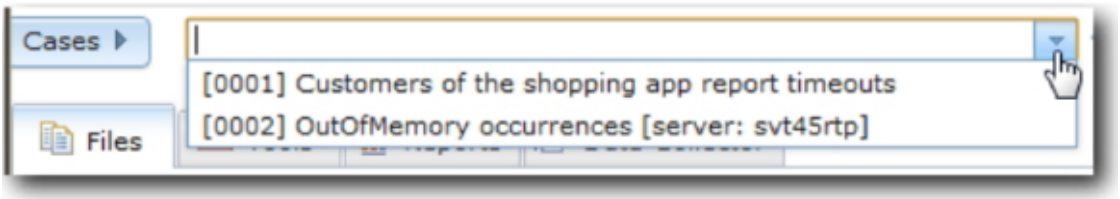
- **Case management**
  Because the Beta 1 release supports a multi-user installation, it is important to have an ability to logically partition your problem determination investigation such that your files, reports, and other artifacts don't intermingle with another team member's investigation artifacts for a separate problem. It will also be critical to separate your artifacts when you investigate several different problems over a long period of time. This enables your diagnostic input files (as well as tooling output files) to be be clearly differentiated. To support efficient organization of related files, IBM Support Assistant supports a basic case concept and provides a Case Management component to help you manage various cases that your team creates within the application.
  A **case** is simply a container for a logical grouping of files and information. A typical practice would be to group artifacts pertaining to a singular issue. For example, if your users are reporting timeouts in a shopping cart application, then you would create a case specifically for diagnosing this reported problem.
  To start creating a container for analysis of this problem, open the **Case Management** panel using the slide out **Cases** tab in the top-left of the UI (Figure 1) and click the **Add** button. IBM Support Assistant will automatically assign a unique identifier for the case; you only need to provide a short summary and description of the problem. The Case Management panel is where you'll perform all of the basic management actions to create, delete, and modify cases.

Figure 1. Case Management panel



Once the case information is provided, you can close the tab and set the case context for your session from the case selection drop-down, as shown in Figure 2. When a case is set for your session, you'll be able to see all the files associated with that case and add new files, such as any Java snapshots, configuration files, or other files relevant to the problem.

Figure 2. Case selection



- **File management**
  You'll work with files associated with a case through the **Files** tab. This is the area where you manage files by performing basic organizational operations against them, such as add, delete, move, and rename. The files that you manage in a case can come from a remote QA, test, or production system, and

could have even been gathered in an automated fashion with a data collection utility like IBM Support Assistant Data Collector.

The Files tab is formatted in a familiar file system explorer-like interface. You can easily navigate through folders in a left navigation panel and see the contents of the folder(s) in the details panel. As you might expect, interacting with files to perform actions against them is possible through a context menu similar to an operating system's file explorer, where items in the context menu trigger actions against the items selected in the files table.

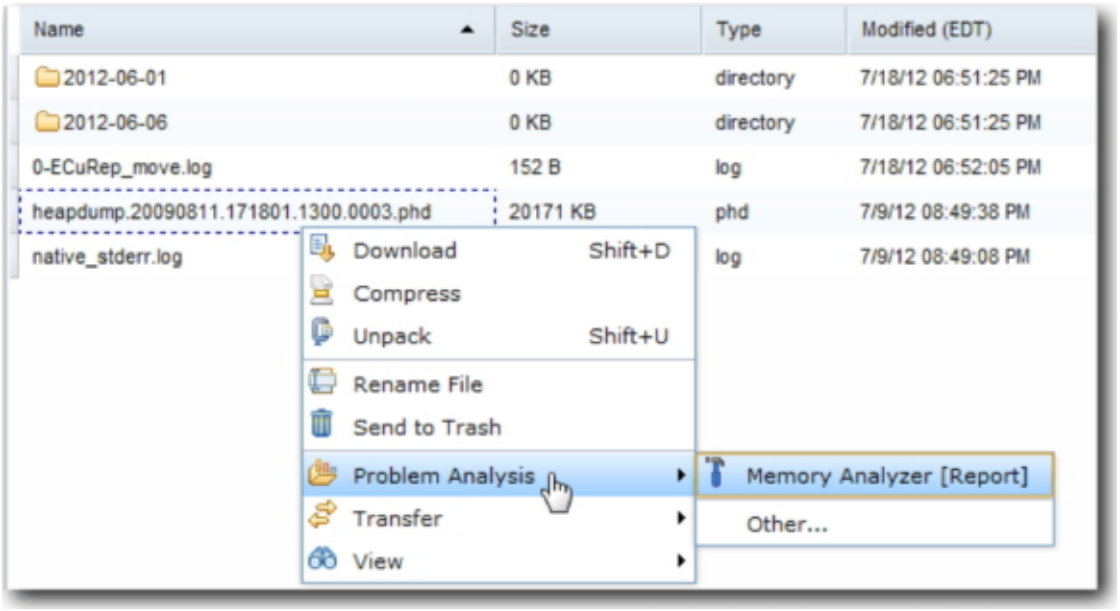
- **File actions**

  There are several common actions available that you can invoke against a file, including:

  - **Download** to your local system.
  - **Compress** and **Unpack archive files**.
  - **Transfer** a file by performing a **Move** or **Copy**.
  - **View** the contents of a file directly in the browser.
  - **Delete** a file by using **Send to Trash**.

  IBM Support Assistant also provides file actions that invoke problem determination tools to kick start your diagnosis. To perform analysis of a file or set of files, you would select the Problem Analysis option from the context menu, as shown in Figure 3. To simplify the selection of an appropriate analysis tool, IBM Support Assistant leverages some file association rules to provide a best-attempt match for a tool that should be used against the selected artifact(s). In Figure 3, you can see that a heapdump*.phd file has been selected in the files table and the Memory Analyzer tool is recommended as a tool that is useful in diagnosing heapdumps. There are actually several tools available in this instance and they're all available from the **Other...** menu option, but IBM Support Assistant has helped narrow down the list of available tools to present the ones that are applicable in a given context.

Figure 3. File actions



Tools

In a Support Analyst role, a primary capability that you'll be interested in getting from IBM Support Assistant is access to tools and utilities that aid you in diagnostics so that you can get at the root cause of a symptom you're investigating. As version 5 evolves, it will grow and focus more on its mission as a problem determination tooling platform and the features that support that direction. IBM Support Assistant provides a variety of types of tools and supporting views and functions to simplify tool understanding and invocation.

In the Beta 1 release of IBM Support Assistant 5.0, a limited number of tools are currently available. While this inventory of tools will likely grow over time, the current tools are good examples of the direction that the application is headed and the types of tools that can be supported.

Types of tools

There are three key types of tools provided in IBM Support Assistant and each type has its benefits and compromises. In some cases, a tool might function as more than one type to support different use cases and needs.

- **Report generator tools**
  Report generator tools accept one or more files or folders as input, then process the data and generate a simple output file, usually in the form of an HTML or .txt report. Other than providing the initial input files and some optional input parameters, these tools are not interactive. These batch tools are simple to run and have the benefit of consuming no local resources. You invoke them

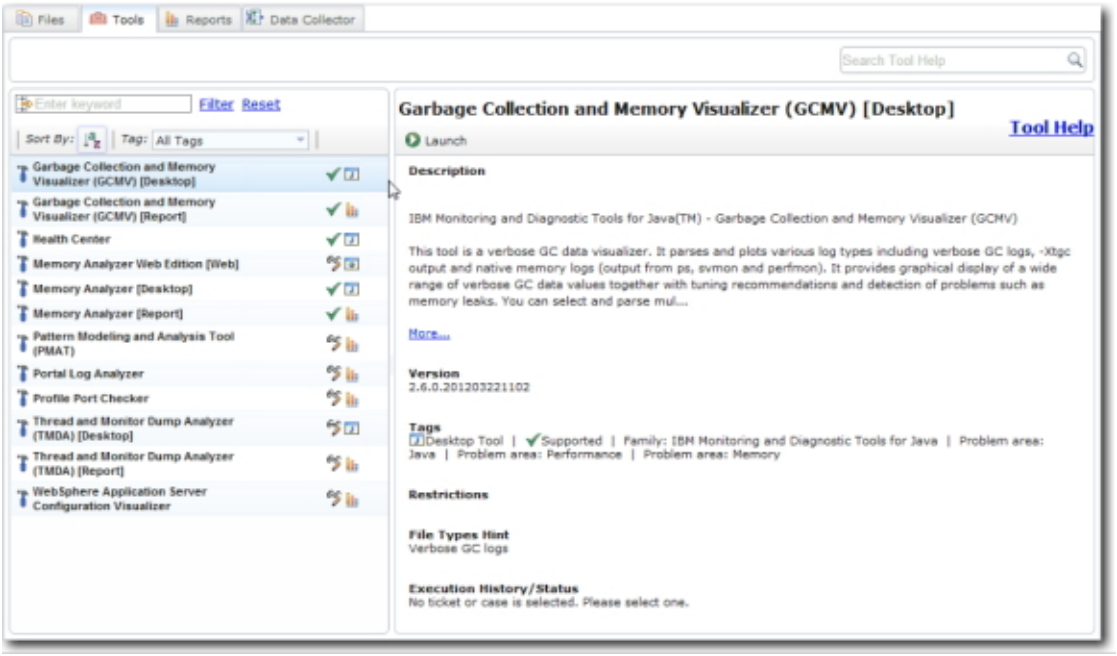
and then can go work on other tasks while waiting for an analysis report to be created.

- **Web-based tools**
  Web-based tools run most of their analysis processing on the IBM Support Assistant server and provide a rich, interactive experience in the UI in the browser. These types of tools are beneficial for activities where you want to offload heavy processing of files to a more powerful server, yet your diagnostic needs demand an interactive experience to perform actions such as drilling into a series of classes.
- **Desktop tools**
  Desktop tools are typical desktop client-side applications that are launched via the IBM Support Assistant browser UI. By leveraging Java WebStart, the entire tool will be installed and run locally on your desktop. This type of tool has a few drawbacks:
  - A Java plugin is required for your browser.
  - Local system resources are required to run the client tool.
  - You must have access to the files you wish to analyze by either downloading them from the application, leveraging mapped drives that enable access to the server file system, or installing IBM Support Assistant locally on your desktop such that the tools and the application are running on the same machine.

Tools catalog

Using the **Tools** tab, shown in Figure 4, you can learn about the capabilities and types of analysis that each tool in IBM Support Assistant can perform, as well as find other information you should know prior to running a tool. You can read about tools, launch them and, for report-generating tools, see the execution status. This is a good place to browse and get familiar with tools.

Figure 4. Tools tab view



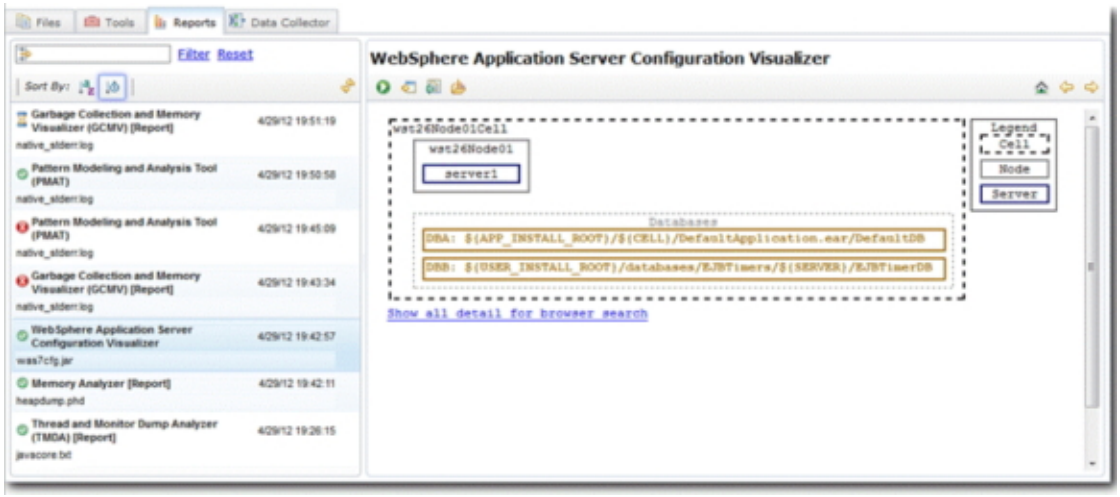
The Tools tab displays a catalog of available tools in a left navigation pane. At a glance, you can learn a little bit about a tool by interpreting the icons displayed. To learn more, you can select it and view the details of a tool in the right panel. This is where you discover the tool's capabilities, understand important attributes that characterize a tool from its tags, and various other attributes. From the details pane, you also have access to a toolbar that enables you to launch the tool and get quick access to the tool's documentation for more detailed usage assistance.

One other key feature to note is a search field in the top-right corner. This field enables you to search for information across all of the available tools' documentation to quickly find information or data you might need. Whether you're searching for information about how to use a tool or just want to discover what tools might be applicable to a particular keyword, this field is a quick shortcut to the full Information Center with tooling documentation.

Reports view

The **Reports** tab provides a quick, concise view into all of the reports that have been generated from the execution of report-generator analysis tools within the current case. Rather than having to browse through the Files tab to locate a tool's output file, you can skim through the list of generated reports in the left navigation (Figure 5) and even quickly tell if the tool succeeded, when it was run, and what its input files were.

Figure 5. Reports tab view



If there's a report you're interested in reading, selecting the entry will display the report directly in the report panel on the right. Further conveniences are available in the toolbar of the reports view to enable you to quickly navigate to the report's input files or output directory. You can re-run the tool if desired, or open it in a separate browser window or tab if you want more screen real estate to read the report.

Available tools

Table 1 shows a list of available tools in IBM Support Assistant 5.0.

Table 1. Available tools

Tool	Type	Description
Garbage Collection and Memory Visualizer (GCMV)	• Report generator • Desktop	IBM Monitoring and Diagnostic Tools for Java - Garbage Collection and Memory Visualizer is a verbose GC data visualizer. It parses and plots J9 and Sovereign verbose GC logs and -Xtgc output. It provides a graphical display of a wide range of verbose GC data values and it handles optthruput, optavgpause, and gencon GC modes. You can select and parse multiple files for comparison.
Health Center	• Desktop	A lightweight tool that monitors active IBM Virtual Machines for Java with minimal performance overhead and provides live tuning recommendations and observations.
Memory Analyzer	• Report generator • Web-based • Desktop	IBM Monitoring and Diagnostic Tools for Java - Memory Analyzer is used to analyze dumps from IBM and Sun Java Virtual Machines to find the causes of various memory problems such as OutOfMemoryErrors.
Pattern Modeling and Analysis Tool for Java Garbage Collector (PMAT)	• Report generator	PMAT parses verbose GC traces and performs pattern modeling analysis of Java heap usage. PMAT produces a report to help you tune the Java heap.

IBM Portal Log Analyzer	• Report generator	IBM Portal Log Analyzer is designed to aid in troubleshooting by using a ConfigTrace.log from Websphere Portal. After executing the tool, view the generated HTML report file in the output directory.
Profile Port Checker	• Report generator	The Profile Port Checker tool scans a collection .jar or .zip file for all server ports. A report is created listing all the server ports that were found along with any duplicate port setting, enabling you to spot potential port conflicts.
Thread and Monitor Dump Analyzer (TMDA)	• Report generator • Desktop	TMDA compares each thread dump and monitor dump and automatically detects hangs, resource contention, Java monitor ownership directional graph structure, and deadlocks.
WebSphere Application Server Configuration Visualizer	• Report generator	WebSphere Application Server Configuration Visualizer generates an interactive HTML visualization of a WebSphere Application Server configuration, including Service Integration Buses and databases. It accepts any combination of .zip or .jar files containing configuration directories. Configurations from multiple nodes in a cell will be merged into a single visualization, and multiple cells can be displayed in the output.

Conclusion

This article presented an overview of the features and functions available in the Beta 1 release of IBM Support Assistant 5.0. These capabilities support a future strategy of delivering a cloud-based architecture for problem determination needs and serve as a solid foundation for the future evolution of troubleshooting tools and services within the IBM software family.

We look forward to providing many new useful functions through future updates to this initial Beta. Feedback on the future strategy and discussion about features (currently available and those desired) is encouraged on the IBM Support Assistant developerWorks forum.

Resources

Learn

- IBM Support Assistant 5.0 Beta home page
- IBM Support Assistant (ISA)
- Enabling Verbose GC logging in WebSphere Application Server
- Enabling dumps in WebSphere Application Server
- Fix Central
- IBM Support Portal
- The Support Authority: If you need help with WebSphere products, there are many ways to get it
- IBM Software product Information Centers
- IBM Software Support Web site
- IBM Education Assistant
- IBM developerWorks
- IBM Redbooks
- WebSphere Software Accelerated Value Program

Get products and technologies

- IBM Software Support Toolbar
- IBM Support Assistant

Discuss

- WebSphere and CICS Support Blog
- Forums and newsgroups
- Java technology Forums
- WebSphere Support Technical Exchange on Facebook
- Global WebSphere Community on WebSphere.org
- Follow IBM Support on Twitter!
    - WebSphere Electronic Support
    - WebSphere Application Server information
    - WebSphere Process Server
    - WebSphere MQ
    - WebSphere Business Process Management
    - WebSphere Business Modeler
    - WebSphere Adapters
    - WebSphere DataPower Appliances
    - WebSphere Commerce
    - IBM Support Assistant Tools

About the authors

Jim McVea

Jim McVea is a technical architect for the IBM Support Assistant project. He joined IBM in 1998 as a Support Analyst and has worked on various support and serviceability initiatives within IBM through the years. Jim's focus continues to be identifying ways to improve the IBM Support Assistant application and analyzing areas to simplify self-help.

Paul Blizniak

Paul Blizniak is a software engineer for the IBM Support Assistant project. His present job has given him much experience with both J2EE and Eclipse technologies. In previous assignments, he has helped develop both the VisualAge Smalltalk and VisualAge for Java products.