IBM® WebSphere® Developer Technical Journal *to go!*

Issue 14.8
December 7, 2011

Inside:

Plus:

IBM WebSphere Developer Technical Journal

Issue 14.8 : December 7, 2011

From the editor

The featured articles in this issue of the **IBM® WebSphere® Developer Technical Journal** are framed around security and performance preparation. Methodically collecting complex integration information at the start of (and throughout) development can go a long way to mitigate serious risk, as can an intelligent approach to implementing OAuth token support with WebSphere Application Server. And you'll want to understand the advantages and trade-offs of moving to a 64-bit JVM for WebSphere Commerce before you do it, and then verify your understanding with informed performance monitoring options. Also included: enhancements to Fix Central that make it easier to use than ever, a methodical solution to virtual image sprawl, and more.

Your required reading begins below...

Feature articles

- **Capturing and analyzing interface characteristics, Part 1: Capturing integration complexity for BPM and SOA solutions**
  by Kim J. Clark and Brian M. Petrini
  Integration still accounts for a majority of the design and implementation effort of many IT solutions. Poorly understood integration requirements can be a significant risk to any project, moreso as the number and complexity of integration points increase. This two-part article describes a procedural technique for capturing information about back-end system integration points, referred to as interface characteristics. You will see how correct analysis of integration points improves estimating and technology selection, and ultimately enables a more predictable pattern-based approach to solution design and implementation. In addition, these articles will specifically highlight the relevance of this approach to enterprise architecture initiatives, such as service-oriented architecture (SOA) and business process management (BPM).

- **64-bit versus 32-bit: Understanding performance implications for WebSphere Commerce sites**
  by Mikhail Genkin
  Now that IBM WebSphere Commerce supports a 64-bit Java Virtual Machine (JVM), you might want to know what the costs and benefits are of using a 64-bit JVM for WebSphere Commerce deployments, when you should make the switch from 32-bit — and if you should switch at all. This article discusses and compares the key differences between the 32-bit and the 64-bit JVMs in terms of their key performance characteristics, costs, advantages, disadvantages, and other practical considerations.

- **Using Trust Association Interceptors with WebSphere Application Server to support OAuth tokens**
  by Lisa Seacat DeLuca, Albert A. DeLucca and Ryan DeJana
  OAuth is an open protocol that permits secure API authorization from web applications (called consumers) by enabling the consumer to act on behalf of a user without requiring the user to provide sensitive account information, such as username and password. If you are looking to share access of your protected resources by becoming an OAuth service provider, this article will help you get started. The solution described here, completed after the authors researched several approaches, uses IBM

WebSphere Application Server (V7.0 or later) with Trust Association Interceptors (TAI) to accept OAuth tokens for authorizing calls from consumers to protected resources. TAIs make it possible to support OAuth alongside other token services, such as LTPA, while meeting WS-Security restrictions. Sample code is included.

- JVM updates in WebSphere Application Server V8:
  Using wsadmin and Jython to easily collect and report WebSphere Application Server PMI data
  by Denis Guillemenot
  IBM WebSphere Application Server provides performance metrics through the Performance Monitoring Infrastructure (PMI), which collects performance and statistical data when WebSphere Application Server is running. Accessing the specific data you want to view usually requires a tool, but it would be convenient if there was also a way to read and understand performance metrics using scripting. Using the popular IBM Tivoli Performance Viewer as a model, this article shows how you can use wsadmin introspection and Jython scripting to process performance data.

The Support Authority

- Major new usability features in Fix Central make it easy and more convenient to find the fixes you need
  by Steve Eaton, Darrin Lemmer and Todd Mitchell
  New dynamic search functions and other exciting new changes have been made to IBM's Electronic Fix Distribution system and its public website, Fix Central. In addition to usability enhancements that were made based on user feedback, a new pilot feature is available that finds the set of installed IBM software products on your server and links you to applicable updates. Find out more in this article.

Comment lines

- Expanding the benefits of proactive monitoring
  by Alexandre Polozoff
  "Is it possible for operations to reduce the mean time to repair (MTTR) and increase the mean time between failures (MTBR), all before users know anything has happened? If monitoring and operations processes have matured to the point that you can stop problems before they impact users and live operations, there must be more that you can do with this capability to fix more than just immediate problems..."

- Defeat image sprawl, once and for all
  by Ruth Willenborg
  "Virtual image sprawl is a reasonably new industry phenomena derived from the simplicity that is the "black box" virtual image. Virtualization and cloud computing make it very easy to create new virtual images, but very hard to know what is in the image and how to manage it. Unfortunately, as image catalogs grow, finding and locating the right images gets harder; new images are created because it is easier to create a new image than it is to figure out what existing image might be reusable..."

Capturing and analyzing interface characteristics, Part 1: Capturing integration complexity for BPM and SOA solutions

Skill Level: Intermediate

Kim J. Clark (kim.clark@uk.ibm.com)
Consulting IT Specialist
IBM

Brian M. Petrini (petrini@us.ibm.com)
Senior IT Architect
IBM

07 Dec 2011

This two-part article describes a well-tested technique for capturing the fundamental interface characteristics of integration points with back end systems. Integration still accounts for a majority of the design and implementation effort of many IT solutions. Poorly understood integration requirements represent a significant risk to a project. You will see how correct analysis of the integration points improves estimating and technology selection, and ultimately enables a more predictable pattern-based approach to solution design and implementation. Moreover, these articles will specifically highlight the relevance of this technique to enterprise architecture initiatives, such as service-oriented architecture (SOA) and business process management (BPM).

Introduction

In terms of importance, one of the most commonly underestimated areas of complexity in an IT solution is the integration with back end systems. Despite the efforts of service-oriented architecture (SOA) to standardize and simplify the way we access back end systems, every new project inevitably brings new points of integration, or, at the very least, enhancements to existing integrations.

This two-part article introduces a way to capture and analyze integration complexity using **interface characteristics** to improve your ability to plan, design, and ultimately implement solutions involving integration.

Part 1 includes:

- **Introducing interface characteristics:** A look at the breadth of information you need to know to truly understand the complexity of an interface.
- **Integration in SOA and BPM:** Why integration is so important to architectural initiatives such as SOA and BPM.
- **Iterative interface analysis:** Because it would be impractical to capture everything about all the interfaces in a solution in one go, this section discusses which characteristics to capture at which stage in the project, and why.

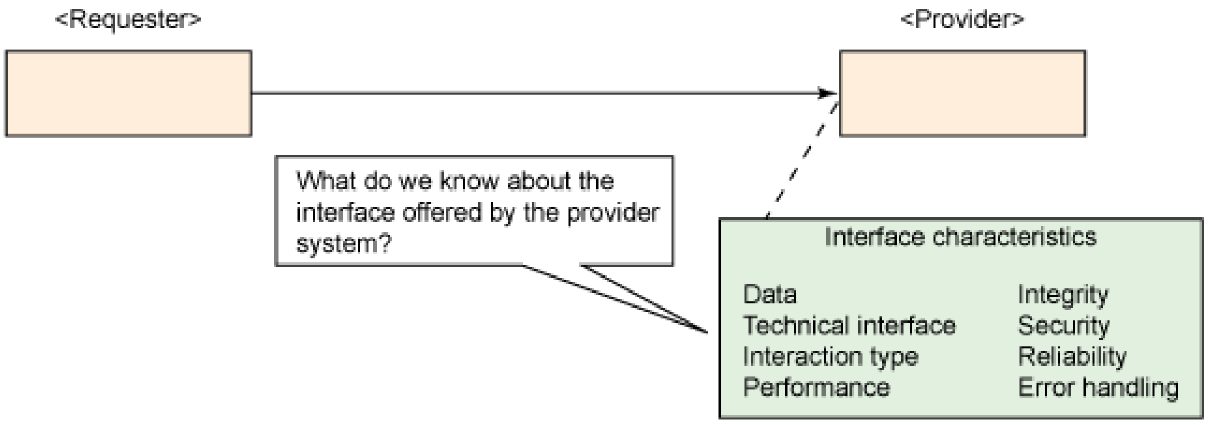
Part 2 will dive into each of the individual interface characteristics in detail.

Introducing interface characteristics

With the tried and tested approach described in these articles for systematically analyzing the integration requirements and capabilities of back end systems, you'll be able to ask just enough of the right questions at each analysis and design stage to ensure that you can effectively estimate, identify risks, and ultimately choose the right design patterns to achieve the desired integration in the solution.

This technique centers on iteratively capturing the key interface characteristics that have the greatest effect on design and implementation. In the simple case of a requester connecting directly to a provider, the interface characteristics are the characteristics of the provider, as shown in Figure 1.

Figure 1. Summary topics of interface characteristics of a provider



Interface characteristics have been presented in summary at many international conferences, and gradually honed on client projects for many years, but this is the first time they have been fully and publicly documented. The full set of key characteristics is shown in Table 1 (these will be described in detail in Part 2).

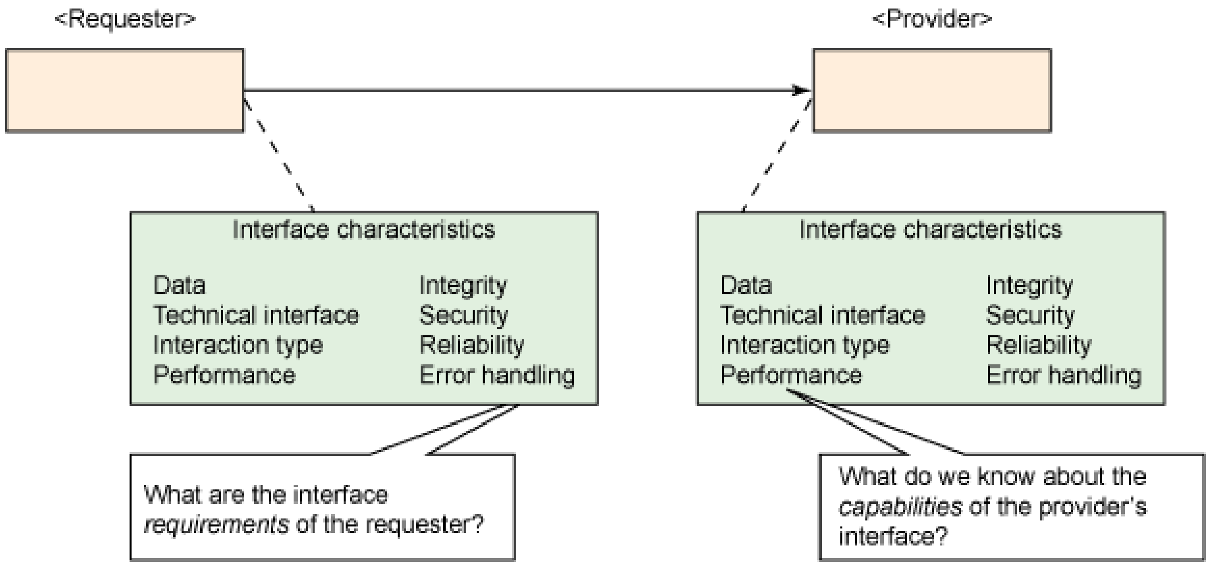
Table 1. The core set of interface characteristics

DATA • Principal data objects • Operation/function • Read or change • Request/response objects	INTEGRITY • Validation • Transactionality • Statefulness • Event sequence • Idempotence
TECHNICAL INTERFACE • Transport • Protocol • Data format	SECURITY • Identity/authentication • Authorization • Data ownership • Privacy
INTERACTION TYPE • Request-response or fire-forget • Thread-blocking or asynchronous • Batch or individual • Message size	RELIABILITY • Availability • Delivery assurance
PERFORMANCE • Response times • Throughput • Volumes • Concurrency	ERROR HANDLING • Error management capabilities • Known exception conditions • Unexpected error presentation

Hopefully, the scope of these characteristics makes it clear right away why it is that many projects fail to assess integration effectively. It is no doubt also clear that it would take significant experience to capture and assess such a large amount of information at one time. Later on, you'll see how this can all be broken down into manageable steps.

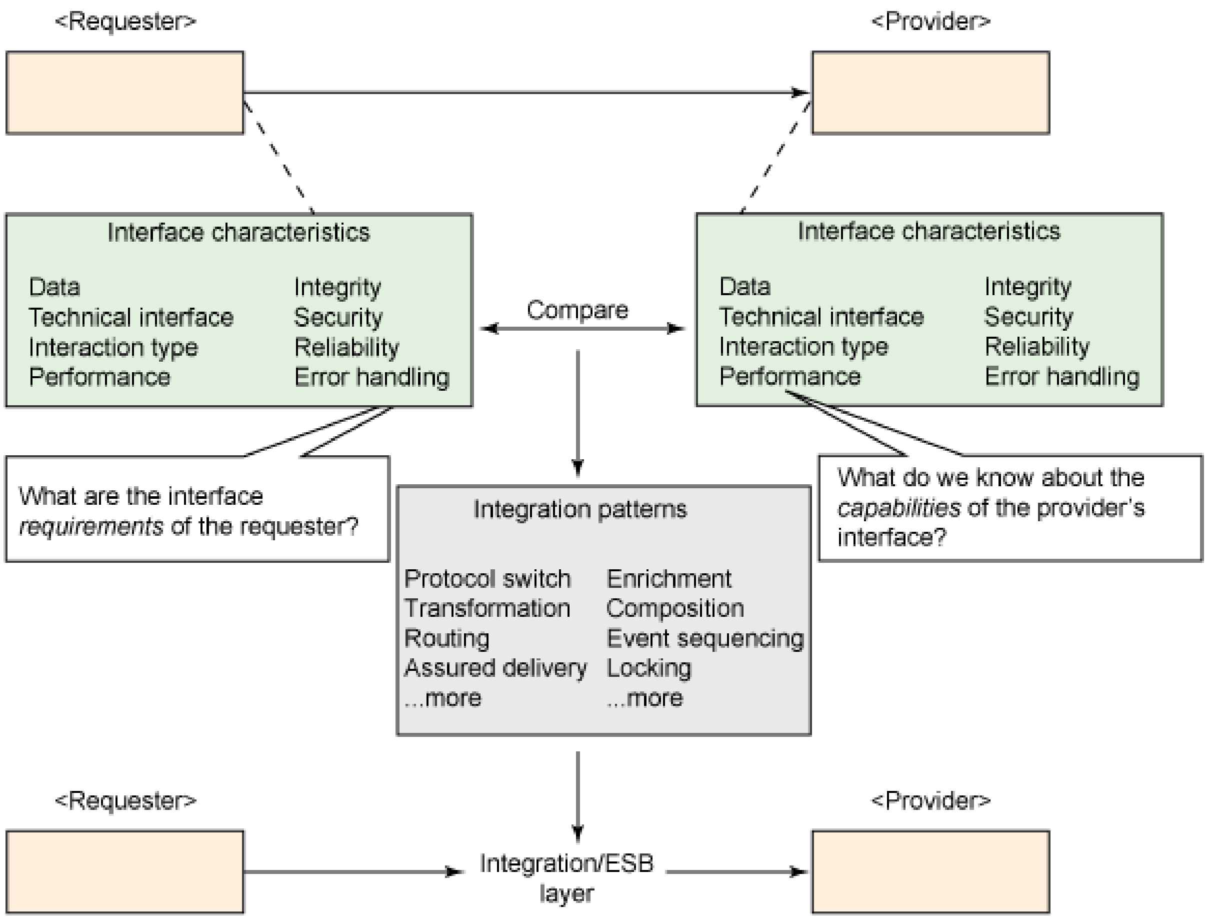
Figure 2 shows that there are really two sets of interface characteristics to be captured. The set on the right includes the capabilities of the provider, but you also need to know the requirements of the requester.

Figure 2. Comparing the gap between the interface characteristics of requester and provider



As you compare the characteristics of requestor and provider, you can then establish the integration patterns that will be required to resolve the differences, as shown in Figure 3.

Figure 3. Examples of integration patterns to resolve differences in interface characteristics



Integration in SOA and BPM

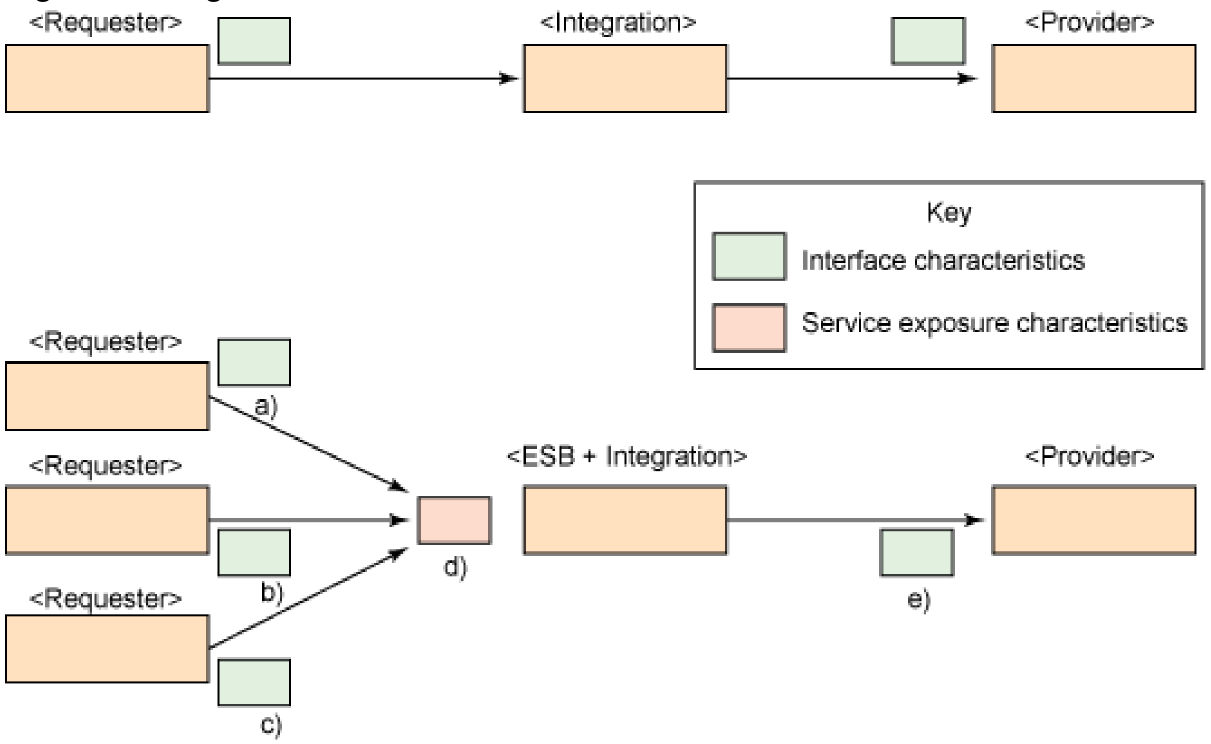
Few projects are executed in complete isolation. Every project is nearly always part of a broader enterprise initiative. Two common initiatives in recent years have been service-oriented architecture (SOA) and business process management (BPM). It is important to understand why interface characteristics are critical to the broader work and how they can be relevantly applied.

SOA

A look at then and now
If you'd like a review, this earlier article on solution design includes a detailed description of how integration architectures have evolved over time.

SOA is a vast topic, but the aspect of it that applies here is where SOA is seen as a logical extension of traditional integration; taking existing integration techniques and then going a step further to make them truly re-usable across a broader set of requesters. The important difference for this discussion is that traditional integration caters to a known set of requestors for which you perform explicit integration. This is shown in the upper example in Figure 4. You do not, therefore, consider the needs of future potential requestors. Patterns such as "hub and spoke" make it easier to introduce new requesters, but each one still requires explicit integration. SOA goes a step further by aiming to assess what the most useful interfaces will be and expose them as services by standardizing they way they are discovered, as shown in the lower part of Figure 4. The objective is that future requesters can simply "use" the service rather than having to "integrate" with it.

Figure 4. Integration vs. SOA



Now, let’s take a look at just how important it is to understand the interfaces in this situation.

To expose services effectively, you need to collate interface characteristics from the anticipated requesters for your system (a and b in Figure 4) and also estimate the potential future requesters (c). You must then compare that with the available interfaces available on providers (e). The hardest part comes next, when you have to use all that information and define the idealized "service" that you could expose for re-use. This is shown in Figure 4 as service exposure characteristics (d). These characteristics for your purposes can be thought of as essentially the same as interface characteristics, except that they are for an exposed service. There are differences, mostly in the sense that many of the characteristics are defined by the governance policies of the SOA, but that detail isn’t important at this level.

Now, imagine if you did that exercise without all of these interface characteristics.

You could get a long way into your design believing that, at a high level, you had a workable solution, only to discover during implementation that some fundamental characteristic completely negates the re-use potential of your service.

Of course, you might not be at the beginning of an SOA initiative. Many services could have already been exposed. If this is the case, you might be able to make some simpler assumptions about how easy it is for your requesters to connect to providers, but you should test our assumptions first. One way to assess this is to understand how mature the SOA is in relation to the service integration maturity model (SIMM) as described by the article referred to above. For example, if interfaces expose SOA-based "governed services," then the enterprise has reached SIMM level 4 (for these interfaces at least) and so you can assume that these interfaces should be easy to re-use. However, do not underestimate just how difficult it is to provide well-governed services. It is almost certain that some of the interfaces will be at a lower level, so it is these that you will need to concentrate on in later phases.

Governed services
To understand what is meant by governed services, see this article, which re-examines terminology and concepts relating to the enterprise service bus (ESB).

Equally important, just because an SOA initiative is taking place, do not assume that all interfaces need to become fully exposed governed services; this would typically be too expensive for most projects. Each will need to be evaluated for its opportunity for re-use. One mechanism you can use to perform this evaluation is the SOMA Litmus Test (see Resources).

BPM

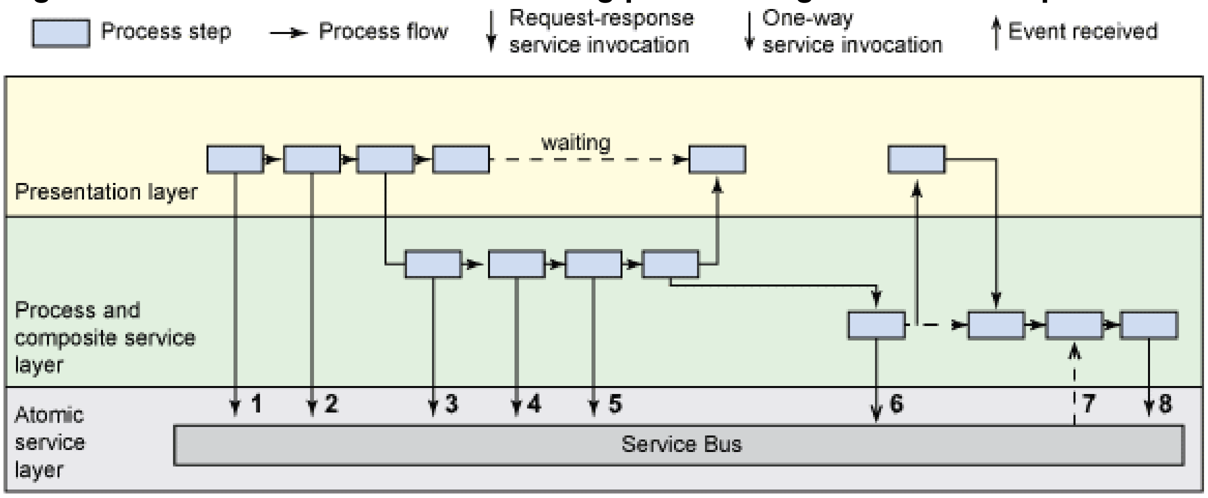
As with SOA, BPM is also a broad topic, but the aspect of BPM that applies here is how business processes interact with back end systems, or providers.

Figure 5 shows that an end to end business process can interact with multiple systems in many different ways. There are a number of things to notice in this diagram.

First, because the process should not, ideally, need to know the details of how to integrate with each of the back end systems individually, the diagram shows only a logical service layer, hiding the detail of the actual integration necessary to get to the back end systems. Therefore, SOA can clearly be complementary to BPM, making integration points needed by the process more easily available.

Second, notice how the type of service requester changes regularly throughout the process. Requests can come from a graphical user interface, from within a brief composition, from a long lived asynchronous process, and so on. Each of these requesters prefers different integration characteristics in the services it calls.

Figure 5. The service interactions taking place during an end to end process



Third, notice how "preferred" integration characteristics vary for different process implementation types. Let’s look at just a selection of the interface characteristic across these interactions, specifically around interaction type and transactionality (the numbers in the list below relate to those that appear in Figure 5):

1. Synchronous request-response non-transactional read
2. Synchronous request-response update non-transactional
3. Synchronous request-response read with transaction lock
4. Synchronous request-response with transactional update
5. Asynchronous fire-forget event
6. Asynchronous event with correlation data
7. Asynchronous event receipt with correlation data
8. Asynchronous request-response update

You can see that each interaction has a preference for certain characteristics. For example:

- A flow through a graphical user interface uses mostly synchronous interactions and does not generally expect to be able to transactionally combine them.
- A synchronous transactional composition might require services that

could participate in a global transaction.

- A long running process saving state over a significant time period might find it easier to interact with asynchronously-exposed services, and will also be comfortable interacting in an event-based style where data is received via completely separate events that contain correlating data.

So, what should you take from this with regard to the relevance of interface characteristics in relation to BPM? Primarily, you should be aware that for what might appear to be the same interaction, the preferred interface characteristics vary considerably depending on the context in which the interface is used. By establishing the most common process implementation types, you might be able to significantly improve the amount of re-use you gain from services you expose by ensuring they exhibit the right characteristics for the common contextual uses.

One final comment on Figure 5, and on BPM in general. Figure 5 is typical of the "swimlane" based process diagrams that are typically used to capture processes for BPM, usually using a notation such as BPMN (business process modeling notation), although a BPMN diagram would not normally show more lanes for the human users of the system and would not directly show the interactions with a service bus. Indeed, Figure 5 is probably already at a more detailed level of granularity than a high level business process should be documented. That pure representation of a business process, completely abstracted from the detail of interactions with back end systems, is an essential part of what makes BPM so powerful as a way of documenting, analyzing, and even implementing business requirements in a rapid and agile way, and is in itself of huge value. However, you must remember just how much this abstraction is hiding when it come to integration. The business process diagram is just the tip of the iceberg when it comes to integration. Clearly, there are circumstances when integration issues might be less of an issue; for example, when most of the business process is human-oriented rather than system-oriented, or if there is a mature SOA on which to build BPM processes, many interactions should be simpler to implement. However, this is not the general case. There are usually many new and complex interfaces -- or different uses of existing ones -- that need detailed rigor and understanding, and you must surface these as early as possible, using interface characteristics to ensure you remove the risk from the overall BPM initiative.

Iterative interface analysis

Because it is impractical (and impossible) to capture all of the integration characteristics in the early stages of a project, let’s look at how this process can be explored iteratively to ensure that just enough information is captured to inform the project and the design at each phase.

Capture of interface characteristics must be aligned with project phases or iterations. There are many different methodologies for defining projects and each use different terminology, but for the sake of simplicity, let's say that regardless of methodology there is always some form of two basic, well-separated exercises:

- **Solutioning**, when you discover in which systems the data can be found, and what types of technical interface are available to those systems.
- **Design**, when you look at the shape and size of the individual data operations that will be required by the solution.

On a traditional waterfall project you would solution the whole landscape based on the project requirements before moving to the next phase of design. In a more iterative or agile methodology, you would solution a relevant slice (often described as a **story**) of the overall problem, then move straight into design and implementation of that isolated slice of the overall picture. Either way, the two phases exist, and so we can discuss here what you should capture at each stage.

Whilst a structured approach is preferred for collecting the minimum characteristics that should be captured at each stage, nothing can replace the eye of an experienced integration specialist, who will be able to infer from the early characteristics captured that deeper investigation into some interfaces will be needed sooner than is suggested here. In the sections that follow, suggestions are included (where possible) regarding what characteristics can be useful ahead of time if they are easily available.

Solutioning

In this phase, you are only aware of the fundamental systems involved in the architecture and the data they will provide to the solution. Early in this phase you will have nothing more than box diagrams on a whiteboard, but by the end you could have the beginnings of an **interface catalogue**.

System context diagram

At this point in the project, you should have created a system context diagram at the very least (Figure 6).

Figure 6. Basic system context diagram

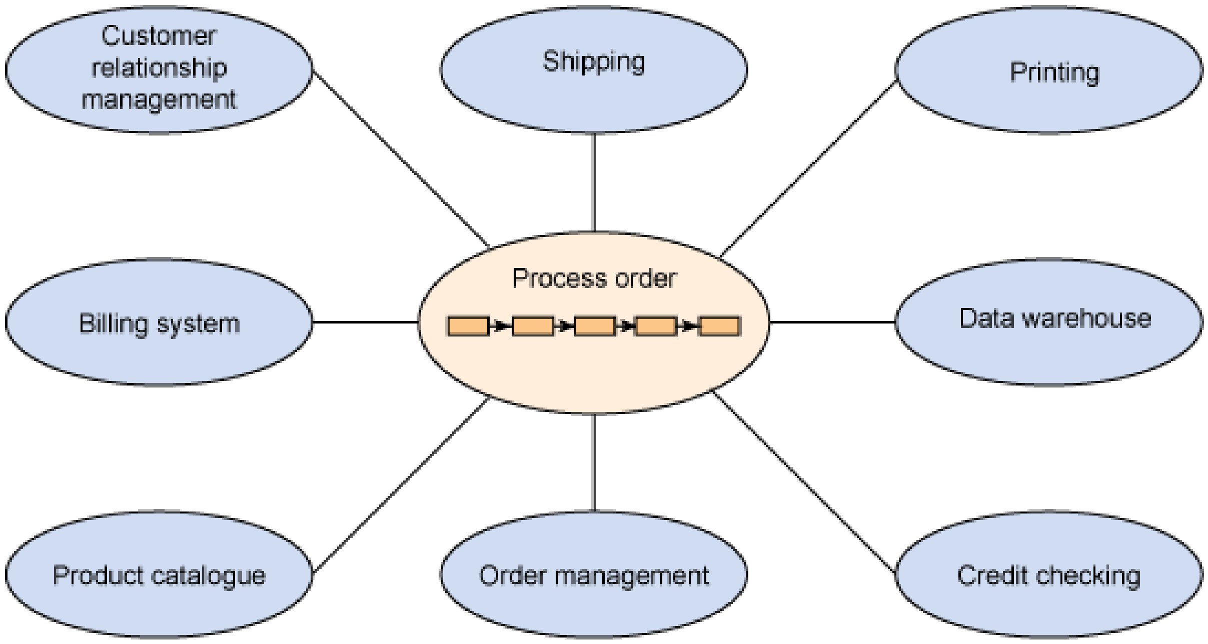
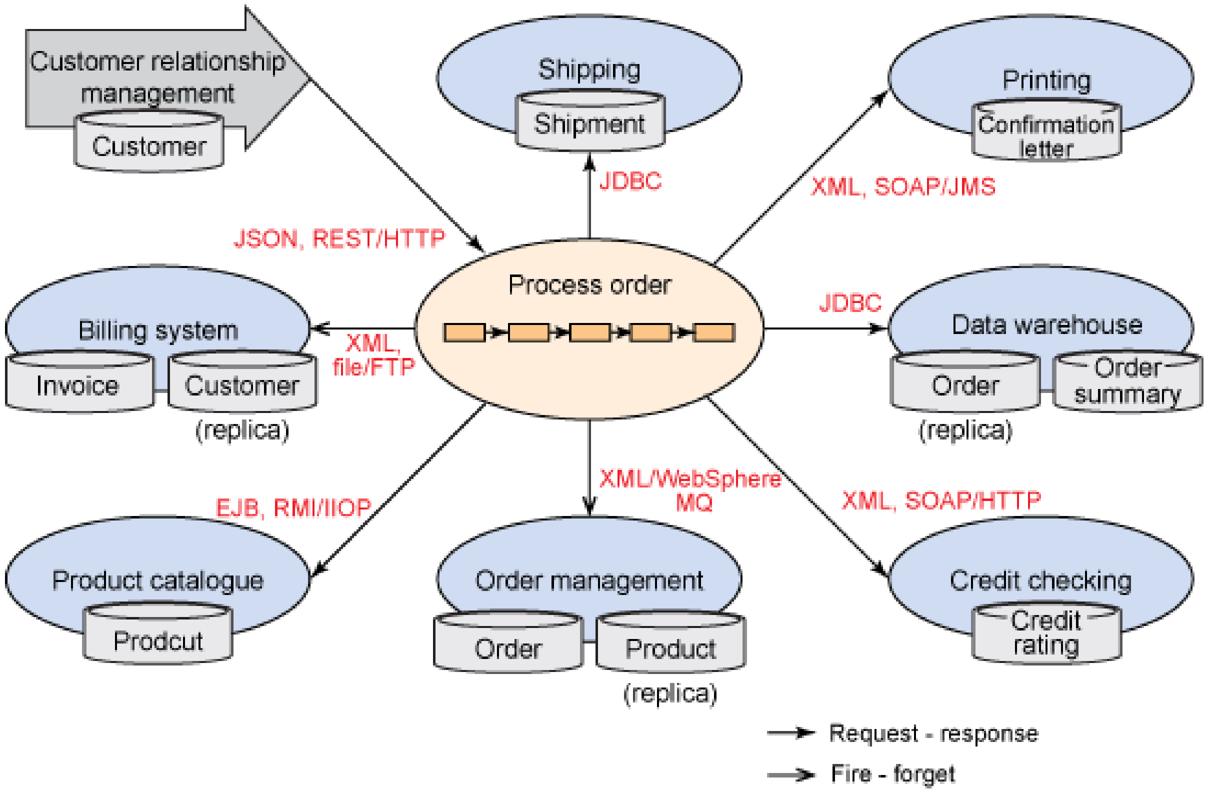


Figure 7 embellishes the context diagram with the available technical interfacing mechanisms (transport, protocol, data format) and principle data objects used by the process to show which system they reside in.

Figure 7. System context diagram embellished with basic characteristics

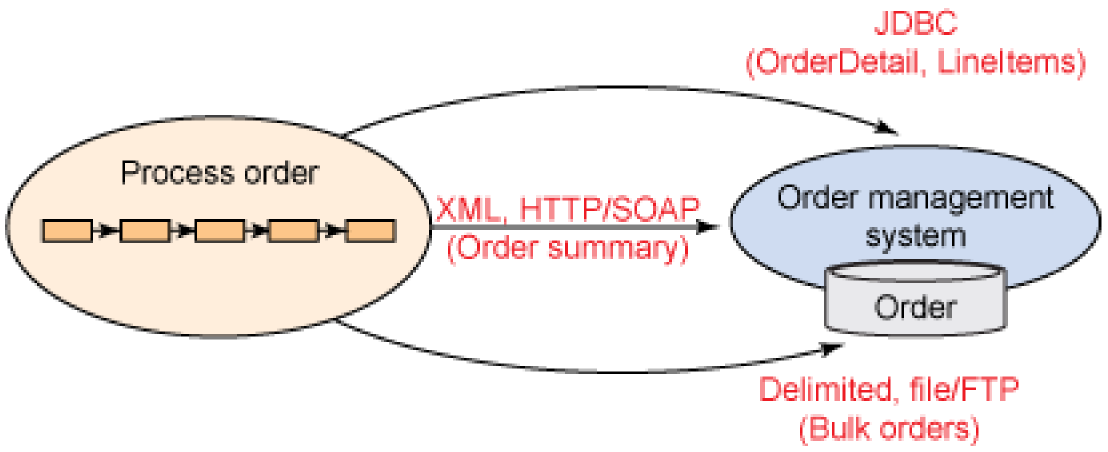


If there are many systems involved, the context diagram might become too cluttered, in which case you might need to capture the characteristics in a separate list. You will eventually need it in list form anyway as you capture more characteristics; this list is generally known as an **interface catalogue**.

Be aware that there could be more than one option for how to interact with a system. You should list them all rather than favor one at this stage (Figure 8). Until you capture the next level of detail, you cannot know which interfaces are appropriate for your use. Indeed, you might use more than one option in a single process. For example, you might do lookups (read) using web services, but use JMS (with assured delivery) for writes.

You would also normally ask about availability at this stage. For example, if a system is down for two hours at night, then a queue might need to be put in front of it to store requests when it is down, rather than perfect a direct JDBC write.

Figure 8. Multiple ways of interfacing with a single system



What do these basic characteristics you have captured tell you, and what more do you need to know?

- **The art of the possible:** You know whether there is an interface with each of the key systems or not. Early identification of systems that have no (or unsuitable) interfaces is a key part of the solutioning process, representing high-risk integration. However, if you have identified characteristics for your interfaces, keep in mind that you could later find that the interface doesn't have all the characteristics that you need, so there is still risk that you might have to build a new interface, or spend time enhancing an existing one.
- **Early research requirements:** You know the core interface technologies you will need to use and thereby what skill sets the project team will need. If any of these technologies are a significant unknown, you can use this early warning to initiate research to improve your knowledge, which will help you in the next phase of the project.

- **Data strategy:** You know which systems your primary data lives in, but, more importantly, you know if it is present in more than one system. If so, you should ensure you understand if there is already a strategy in place to keep data in synchronization, or whether you will need to put that in place. Put another way, you need to identify which system is the master for each data item (the single version of the truth), and if there is more than one, how conflicting updates will be handled.

In short, you have improved your understanding of the complexity and reduced the risk -- but certainly not eliminated it. The risk could still be high, but you at least know where it lies. You must move a level deeper if you are to have any confidence in your estimates at this stage.

Gap analysis for risk assessment and estimating

At the solutioning stage, you will still likely need to provide an assessment of the risk involved in the project and some high level estimates. The information you have so far is simply not enough to enable you to do that.

Table 2. Interface characteristics during solutioning

DATA • Principal data objects **- Required** • Operation/function • Read or change • Request/response objects	INTEGRITY • Validation • Transactionality **- Optional** • Statefulness • Event sequence • Idempotence
TECHNICAL INTERFACE • Transport **- Required** • Protocol **- Required** • Data format **- Required**	SECURITY • Identity/authentication • Authorization • Data ownership **- Required** • Privacy
INTERACTION TYPE • Request-response or fire-forget **- Optional** • Thread-blocking or asynchronous • Batch or individual **- Optional** • Message size	RELIABILITY • Availability **- Optional** • Delivery assurance
PERFORMANCE	ERROR HANDLING

• Response times **- Optional**	• Error management capabilities
• Throughput **- Optional**	• Known exception conditions
• Volumes **- Optional**	• Unexpected error presentation
• Concurrency **- Optional**	

The initial characteristics you captured on the solution context diagram (shown as **Required** in Table 2) refer only to the mechanisms by which you can exchange data with the systems. You are not yet at the level of the individual functions or operations that can be performed through that interface. For example, you might only have an understanding of the overall volumes of data that will be passed over an interface, but not the request rates for each specific data exchange. Even with this, you would be able to establish if the interface will be completely overwhelmed by the new requirements. You will reach the full level of detail in the next phase, but there are still many early warning signals you could get if an experienced integration specialist were to consider (even at a high level) a slightly deeper set of characteristics. A suggested set of further characteristics to look into are shown as **Optional** in Table 2.

You might be wondering where the detail of the *overall volumes of data to be passed over the interface* came from. This required not only knowledge of the interface exposed by the interface, but also an understanding of the requirements of the requester. At this stage, you need to do more than simply establish what interfaces the systems make available. As you saw back in Figure 2, you also need to capture the requirements of those that will make requests on those systems.

From here on, your capture is all about comparing the differences between requirements of the requestor and the reality offered by the providers. These differences will enable you to understand the integration complexities you will have to overcome. For example, if you know that the requester will require real-time access to data, but the only interface available is batch-based, you already know that interface will probably not be adequate. If there is no other interface currently available, you will have to design and implement a completely new one. You should assume this integration problem will be high risk and complex. On the other hand, you might be fortunate that there is already a fully governed SOA service available for re-use that you can use. In theory, this should significantly lower the risk and complexity of the integration although, as noted earlier, you need to consider how mature and well governed the SOA is before making any assumptions.

Finally, along with the additional characteristics noted above, you might also want to get a rough understanding of the granularity of the interfaces compared to what is required. This is difficult, as you will not gain a detailed picture of this until the next phase. However, composition adds significant complexity to the integration, so it is a critical factor.

Since you cannot yet build an estimate on these additional factors in detail, the typical approach is to delve deeply into a representative sample set of interactions and extrapolate from there. These also provide opportunity for risk mitigation, as they can be used for proof of concept exercises.

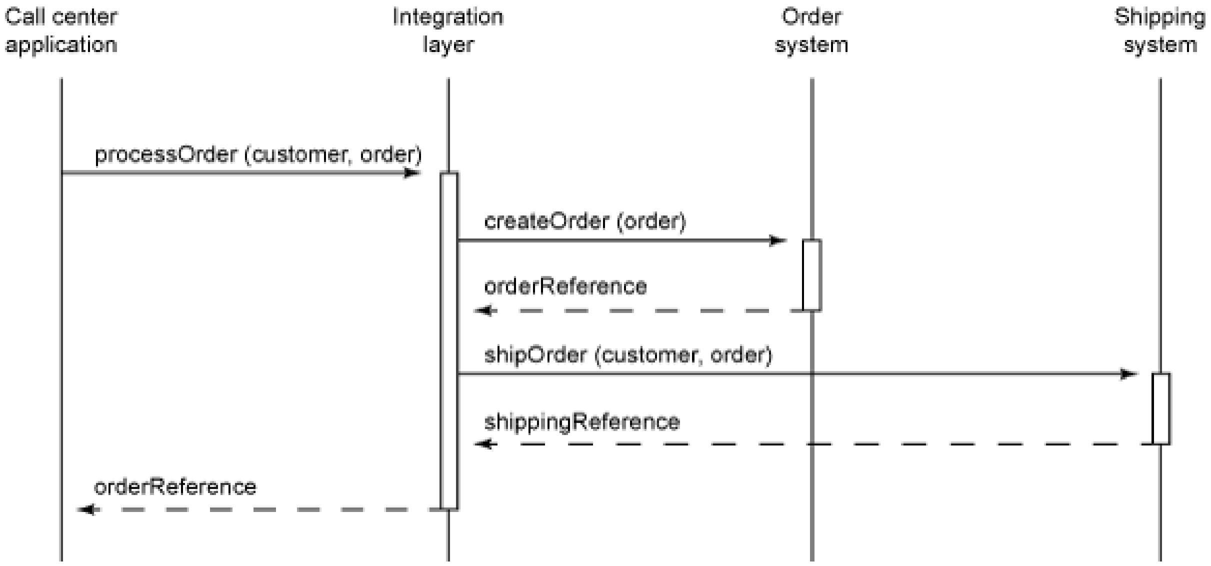
Design

Individual interaction design

In the previous phase, you had no time to establish the specifics of each system interaction, except for perhaps the representative examples created for estimating.

In this phase of the project, you will be realizing the specific functional interactions of the solution by building out an understanding of the specific data exchanges that will take place across the interfaces you already defined in the design phase. These exchanges are typically represented in some form of sequence interaction diagram, such as that shown in Figure 9.

Figure 9. Sequence interaction diagram showing specific method signatures and composition in the integration layer



The sequence diagrams force you to establish whether the systems’ interfaces provide the specific data operations that you require. This implicitly populates the remaining characteristics in the Data section. It is important that you use the same gap analysis technique you used earlier, understanding the problem from the requester’s point of view, and then consider what the provider has to offer.

Interface granularity and composition

There is some key information that will spill out from the creation of sequence diagrams – whether or not the granularity of the available interfaces is correct. It is often the case that the requester has a more course-grained functional requirement than the provider’s interface. For example, the requester requires a customer, with any current orders, and a list of the high level details of any accounts held. This might require several requests to a provider, and maybe even to multiple providers. Figure 9 shows a further example of a mismatch in granularity whereby order creation and shipping need to be combined into a single operation to "process the order." It’s easy to see why the sequence diagrams are so important at this stage

This difference in granularity means you will need to perform composition (multiple requests wrapped up as one). The macro design must establish where this composition should take place: in the requester, in the integration layer, or in the back end system itself. The decision will be based on a balance between factors such as performance and re-use. (Where and how this composition should take place is beyond the scope of this article.)

What is relevant to the core concept of this article, however, is just how important the other characteristics become if composition is required. Here are some common examples:

- If a composition requires more than one interaction that **changes data**, you need to consider whether those interactions can be **transactionally** combined, and, if not, how you will perform **error management** if the request fails between the updates.
- If the individual requests of a composition need to be performed in parallel in order to meet the **response time** requirements, do you have sufficient **concurrency** to handle the increased number of connections required. And if the system interfaces are **thread-blocking**, how will you spawn and control the additional threads required?
- If a composition interacts with multiple systems, how will you retain the separate **security** information and any other **stateful** context such as sessions and connections required?

Establishing the integration patterns

Once you know the specific operations that will be performed, you can then explore all the other characteristics in the proper detail for each operation, both in terms of the requestor’s requirements and the existing interface’s current capabilities.

As you compare the characteristics of requestor and provider, you can then establish the integration patterns that will be required to resolve the differences, as you saw in Figure 3. Be aware that the "composition" we have just been discussing is itself one of the integration patterns.

At this stage, the patterns are defined at a logical level, without reference to the specific technology. However, as commonly used integration patterns are well established, standard implementation techniques within the available technologies should also be explored, tested, and documented. This will significantly streamline the micro design phase, next.

Micro design

In the micro design phase, you use the captured characteristics to complete the technology-specific aspects of the design.

Notice that all the interface characteristics are agnostic of the integration technologies; they are logical characteristics. Therefore, capturing interface characteristics should have been **completed** by the time you reach micro design, and you are purely **using** the characteristics to aid you in the final details of the design. One of the most common integration-related causes of projects running over budget is ineffective capture of interface characteristics **prior** to micro design, leaving inestimable -- possibly unresolvable -- issues for this phase, or, worse still, for implementation. By far the most common and most troublesome issue here is failing to assess and capture a sufficient amount of the **data** characteristics prior to micro design.

Beware of good intentions that aim to "add that on later," especially for characteristics that are cross-cutting concerns, such as security. It is acceptable to defer something only once you’ve established how complex it is, and how much harder it will be to add it onto a running implementation.

The primary activities in this phase that make use of the interface characteristics are:

- Bringing precision to the data model, such as defining of detailed data types, and restructuring of data required for physical representation.
- Detailing the technology specific implementation of the integration; what technologies features will be used, and how.
- Designing for performance, using the performance characteristics to assist in choosing between multiple technology options.

Conclusion

This article described a mechanical way to analyze integration requirements using interface characteristics, and discussed why this is of such critical importance, not only to traditional enterprise application integration but to other enterprise initiatives such as SOA and BPM as well. This article also looked at how this technique can be used iteratively across the lifecycle of a project to ensure you have the right information at the right time.

Part 2 will look at each individual interface characteristic in detail to help you acquire a clear understanding of their meaning and importance.

Acknowledgements

The authors would like to thank the following for their contributions to and reviews of this paper: Andy Garratt, David George, Geoff Hambrick, Carlo Marcoli.

Resources

- Enterprise Service Bus, re-examined
- Increase flexibility with the Service Integration Maturity Model (SIMM)
- Solution design in WebSphere Process Server: Part 1
- Patterns and Best Practices for Enterprise Integration
- Executing SOA: A Methodology for Service Modeling and Design
- Enterprise Connectivity Patterns: Implementing integration solutions with IBM's Enterprise Service Bus products
- Solution design in WebSphere Process Server and WebSphere ESB, Part 3: Process implementation types: Patterns based design for process-based solutions
- developerWorks BPM zone: Get the latest technical resources on IBM BPM solutions, including downloads, demos, articles, tutorials, events, webcasts, and more.
- IBM BPM Journal: Get the latest articles and columns on BPM solutions in this quarterly journal, also available in both Kindle and PDF versions.
- IBM developerWorks WebSphere

About the authors

Kim J. Clark

Kim Clark is an IT Specialist from the United Kingdom working in IBM Software Services for WebSphere (ISSW). Alongside providing guidance to customers he writes and presents regularly on SOA design. He has been working in the IT industry since 1993 spanning object oriented programming, enterprise application integration (EAI), and SOA. He pioneered many of the early projects using SOA Foundation Suite products. Kim holds a degree in Physics from the University of London, England.

Brian M. Petrini

Brian Petrini is a Senior IT Architect on the IBM Software Services for WebSphere consulting team. He takes part in the Focused Technology practice for IBM WebSphere Business Integration.

64-bit versus 32-bit JVM: Understanding performance implications for WebSphere Commerce sites

Skill Level: Intermediate

Mikhail Genkin (genkin@ca.ibm.com)
Performance Architect, WebSphere Commerce
IBM

07 Dec 2011

Starting with Version 7 Fix Pack 1, IBM® WebSphere® Commerce supports a 64-bit Java™ Virtual Machine (JVM). This article discusses the costs and benefits of using a 64-bit JVM for WebSphere Commerce deployments, a typical memory utilization, and performance characteristics. When to switch and what to consider when moving to a 64-bit JVM is also discussed.

Introduction

IBM WebSphere Application Server comes in two flavors – 32-bit and 64-bit. More specifically, you have the option of using the 32-bit or the 64-bit Java Virtual Machine (JVM) with your WebSphere Application Server. Prior to Version 7 Fix Pack 1, IBM WebSphere Commerce supported only the 32-bit version of WebSphere Application Server. Starting with V7 FP1, WebSphere Commerce added support for 64-bit.

This article discusses the key difference between how the 32-bit and the 64-bit JVM utilize memory when serving a typical WebSphere Commerce workload. It also discusses how these JVMs compare in terms of their key performance characteristics – throughput, response time, and overall memory utilization. It concludes with a discussion on what you need to consider when moving to a 64-bit JVM and how this affects your planned deployments.

Total memory equation in WebSphere Commerce deployments

WebSphere Commerce is a sophisticated application that creates many different types of Java objects. When a shopper enters a WebSphere Commerce site, he first interacts with JSPs, JSP fragments, and servlets. The JSPs invoke servlets, which in turn invoke session beans (EJBs) and databeans.

The session bean layer typically invokes controller commands, which invoke task commands. These task commands invoke entity beans (CMP EJBs). The entity beans execute SQL statements against the database, fetch result sets, and store relevant data in their properties.

Session beans, controller commands, and task commands execute business logic that uses Java variables, constants, objects, and collection classes to store transient data on the stack and in the JVM heap. These data are not persisted, but they have a footprint on the overall memory utilization of your site.

These objects fall into one of two categories:

- **JVM-bound objects**
  This category represents objects that cannot be cached externally to the WebSphere Commerce JVM and includes:
  - Session beans.
  - Entity beans.
  - Access beans.
  - Stack variables.
  - Transient business logic variables allocated on the heap.
  - Objects returned by external systems (for example, external search engines) for processing and displaying by WebSphere Commerce.
- **Cacheable objects**
  These objects can be cached externally to the WebSphere Commerce JVM by utilizing either the dynacache disk offload feature, edge caching, or WebSphere eXtreme Scale. These objects include:
  - JSPs.
  - JSP fragments.
  - Servlets.

- Controller commands.
- Task commands.
- Java objects that extend the DistributedMap interface.
- Contents of the WebSphere Commerce data cache.
- Contents of the WebSphere Commerce marketing cache.

Note: At this stage, some of the JVM-bound objects can also be cached. For example, JDBC result sets are cached in the statement cache provided by the WebSphere Application Server connection pool. Entity beans are cached in the EJB cache. Know that these objects cannot be stored externally to the JVM. If the total memory space required by these objects is larger than what the JVM can provide, the site will experience a performance bottleneck.

If you take a snapshot of the WebSphere Commerce JVM heap, you see that JVM-bound objects typically do not take up more than 200 to 300 megabytes. The remainder of the JVM heap is occupied by cacheable objects that are retrieved and loaded into the JVM heap as needed. This difference between JVM-bound and cacheable objects is key to understanding performance between a 32-bit and a 64-bit JVM.

The 64-bit JVM

WebSphere Commerce is a J2EE™ application that is deployed on WebSphere Application Server, which again comes in two editions: 32-bit and 64-bit. These two editions of WebSphere Application Server are identical in every way except one: they include different editions of the IBM J9 JVM (except on the Solaris™ platform, where they include different flavors of Sun's™ JVM).

The IBM J9 JVM comes in two editions: 32-bit and 64-bit. These two editions are identical to each other in every respect except one: how they address memory. The 32-bit edition has a limit on the maximum heap size that is addressed by the JVM. This limit varies by platform, but it is generally about 1.5 GB.

The 64-bit JVM can address a much larger heap size than the 32-bit JVM. Version 1.6 of the IBM J9 JVM introduced compressed references technology. Processing larger 64-bit references does come at a cost, and compressed references reduce this cost and make it more reasonable. The impact of compressed references is described in more detail later in the article. The concept is introduced here because it affects the maximum heap size addressed by the JVM.

With compressed references on (they are on by default), the IBM J9 JVM can address about 28 GB of memory. Without compressed references, the maximum heap size is unlimited.

One important caveat to keep in mind is that, currently, WebSphere Application Server does not provide in-version migration support for migrating from a 32-bit to a 64-bit JVM. If you start by deploying one edition of WebSphere Application Server and then decide to switch to the other, a re-installation is required. Because of this fact, it is important to understand the trade-offs between 32-bit and 64-bit performance, and make the right choice early in the project life cycle. The next section discusses how WebSphere Commerce performs on a 64-bit and a 32-bit JVM.

Comparing 64-bit and 32-bit performance

How do you go about comparing 64-bit with 32-bit performance? For WebSphere Commerce, this comparison involves comparing the cost of using larger references to the cost of out-of-process memory access. This sounds complicated at first glance, but in the case of WebSphere Commerce, it is actually fairly straightforward.

Virtually all WebSphere Commerce sites make extensive use of caching. The typical size of a WebSphere Commerce cache is 5 to 6 gigabytes (GB). The 32-bit JVM can hold only about 1 GB of this in Java heap memory. WebSphere Application Server dynacache provides a feature called **disk offload**. This feature enables those cached objects that do not fit into the JVM heap to be offloaded to the disk. However, this does not mean that those objects have to be always retrieved from the disk. All operating systems supported by WebSphere Commerce provide a file-system cache capability.

When enabled and appropriately sized, the file system cache ensures that the disk offload file containing objects that do not fit into the JVM heap is also served out of RAM. Essentially, for a well-tuned WebSphere Commerce site, the trade-off between running a 64-bit versus 32-bit JVM involves in-process versus out-of-process memory access.

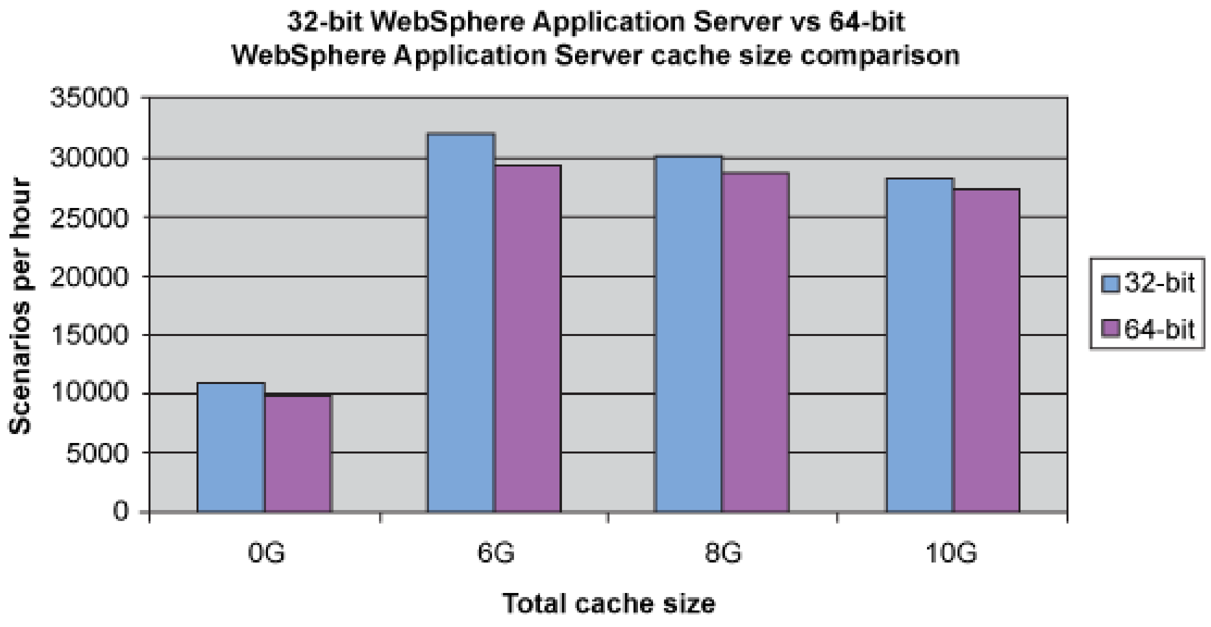
Comparing throughput characteristics

Before delving deeper into performance data, keep in mind that performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance for any user varies depending upon many factors, including the amount of customization, the I/O configuration of the system, the storage configuration, and the workload processed. Therefore, there is no assurance that an individual user will achieve results similar to those stated here.

To make a fair comparison of a 64-bit versus a 32-bit JVM throughput characteristics for a WebSphere Commerce workload, you need to consider the total size of the cache. As discussed above, the heap size of the running WebSphere Commerce JVM is largely occupied by cacheable objects. In the case of a 32-bit JVM, most of these cacheable objects that do not fit into the JVM heap need to be offloaded to disk, and end up in RAM via the file system cache. In the case of a 64-bit JVM, all cacheable objects can fit into a much larger heap size.

Figure 1 shows a throughput comparison for a 32-bit and a 64-bit JVM running a WebSphere Commerce workload. In this study, we varied the total size of the cache from 0 GB (no caching at all) to 10 GB. In all cases, the system had sufficient RAM to fully contain the file system cache, including the dynacache disk offload file. For each cache size, a step-up test was performed to establish the maximum possible throughput. In all cases, the 32-bit JVM slightly outperformed the 64-bit JVM. Although the compressed references technology available in the latest version of the 64-bit JVM has reduced the overhead of processing larger references, you still see a 5% to 15% penalty.

Figure 1. Throughput comparison for different WebSphere Commerce cache sizes



Effect of the file system cache

Figure 2 shows additional data points, comparing relative throughput performance for the 32-bit and 64-bit JVM. Look closely at the one showing throughput comparison for the large 30 GB total cache size. In this case, when we reduced the size of the file system cache available to the 32-bit JVM, its throughput performance declined significantly when compared to the 64-bit JVM.

Figure 2. Effect of the file system cache on 32-bit JVM throughput

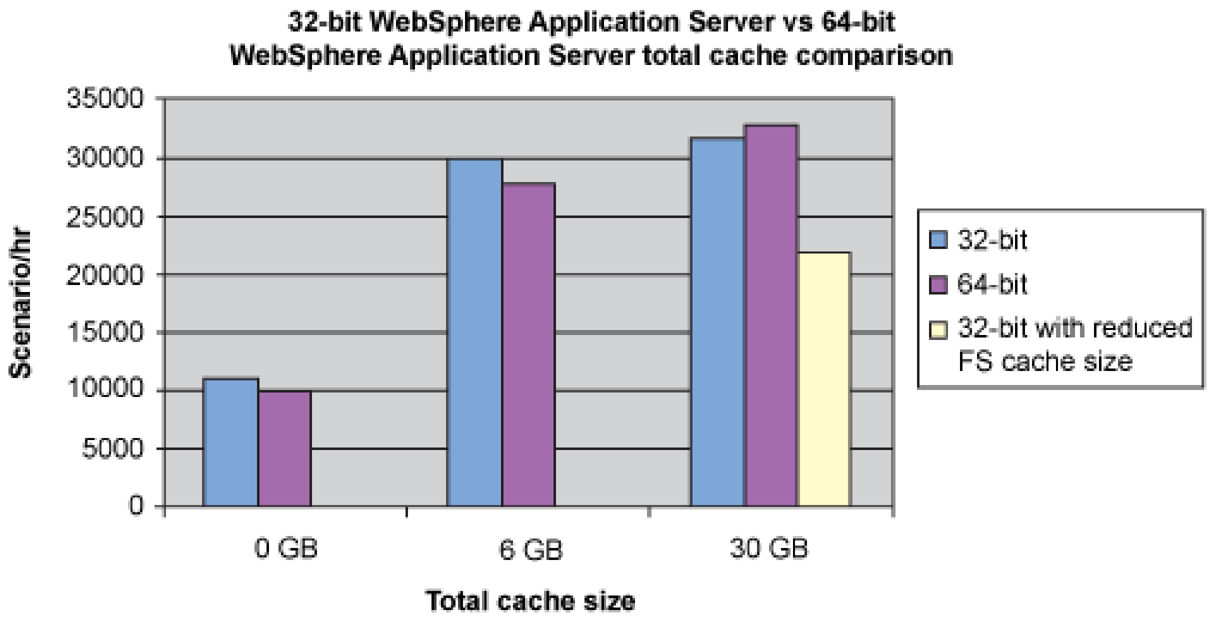


Table 1 shows that the total memory footprint for the 32-bit system is about 7.9 GB. This is determined by adding the actual heap utilization and the size of the disk offload file. In the case of the 64-bit JVM, all of the memory utilized was in the Java heap, and the size of this heap was about 10 GB. As you can see, the 64-bit JVM running a typical WebSphere Commerce workload tends to utilize about 30% more memory than an equivalent 32-bit system running the same workload. The garbage collection overhead for the two systems tends to be about the same.

Memory footprint and garbage collection overhead

Table 1 compares the memory utilization and garbage collection (GC) overhead observed during one of our runs.

Table 1. Comparison of total memory utilization and garbage collection overhead

	32-bit	64-bit
Actual heap size	1.2 GB	~10 GB
Disk offload file Size	~6.7 GB	0
GC overhead	~8% (max 25%)	~9% (max 13%)

For the 32-bit system, the total memory footprint is determined by adding the actual heap utilization and the size of the disk offload file. In this case, it is about 7.9 GB. In the case of the 64-bit JVM, all of the memory utilized was in the Java heap. The size of this heap was about 10 GB. As you can see, the 64-bit JVM running a typical WebSphere Commerce workload tends to utilize about 30% more memory than an equivalent 32-bit system running the same workload. The garbage collection overhead for the two systems tends to be about the same.

Moving to 64-bit

One of most frequently asked questions that we get from customers is: *Now that WebSphere Commerce supports 64-bit, should I move to a 64-bit JVM?* The answer is that this depends on a number of factors, and not all of them are strictly performance-related. Our lab testing indicates that a well-tuned 32-bit deployment still outperforms an equivalent 64-bit deployment. However, the 32-bit deployment requires more tuning to achieve this.

To achieve top performance from a 32-bit deployment, you need to be concerned with (among other tunings):

- JVM tuning.
- Dynacache disk offload settings.
- Size of disk offload files (one for each JVM deployed on the server).
- Efficiency of the file system cache (which varies on different platforms).
- I/O performance of disk that are used (typically a Storage Area Network).

In contrast to this, for the 64-bit deployment, you only need to make sure that you have sufficient RAM to hold the entire heap (including cached data) and tune the JVM. Tuning the 64-bit JVM requires the same steps and overall approach as the 32-bit JVM, except that key settings, such as the starting and maximum nursery size and starting and maximum heap size, all need to be proportionately larger.

Another important consideration is that not all heap size problems encountered by customers are necessarily related to the total size of cacheable objects that need to be contained. In some cases, the total size of non-cacheable, JVM-bound objects produced by for example, integration and customization, can become too large to fit into the 32-bit heap. In this case, the 64-bit JVM allows you to simply increase your heap size, potentially avoiding (or at least postponing) expensive code re-design and re-factoring exercises.

Conclusion

For high-volume WebSphere Commerce customers who want top performance from their hardware, the 32-bit JVM is still the recommended choice. However, the 64-bit JVM is now fully supported by WebSphere Commerce, and it does provide a number of advantages:

- Reduced I/O traffic to a Storage Area Network (SAN).

- Simpler overall tuning.
- Greater flexibility when dealing with heap contention caused by an accumulation of JVM-bound objects.

For many customers, these benefits may be well worth the price of a fairly reasonable performance overhead associated with using the 64-bit JVM for WebSphere Commerce deployments.

Resources

Learn

- WebSphere Application Server V7 64-bit performance
- Top 10 64-bit IBM WebSphere Application Server FAQs (PDF)
- WebSphere Commerce V7 Information Center: JVM performance tuning (advantages of the 64-bit JVM)
- WebSphere Commerce V7 Information Center
- developerWorks WebSphere Commerce zone

Discuss

- WebSphere Commerce discussion forum

About the author

Mikhail Genkin
Mikhail Genkin is the Performance Architect for WebSphere Commerce. He has 15 years of software development experience and has contributed to many IBM products including WebSphere Commerce, WebSphere Process Server, WebSphere Integration Developer, Rational Application Developer, and Visual Age for Java-Enterprise Edition.

Using Trust Association Interceptors with WebSphere Application Server to support OAuth tokens

Skill Level: Intermediate

Lisa Seacat DeLuca (ldeluca@us.ibm.com)
Advisory Software Engineer
IBM

Albert A DeLucca (aadelucc@us.ibm.com)
Senior Software Engineer
IBM

Ryan DeJana (rdejana@us.ibm.com)
IBM Certified IT Specialist
IBM

07 Dec 2011

Are you looking to share access of your protected resources by becoming an OAuth service provider? This article describes how you can use IBM® WebSphere® Application Server (V7.0 and later) with Trust Association Interceptors (TAI) to accept OAuth tokens for authorizing calls from applications or web sites (consumer) to protected resources. TAIs make it possible to support OAuth alongside other token services, such as LTPA, while meeting WS-Security restrictions.

Introduction

OAuth (Open Authorization) is an open protocol that permits secure API authorization from desktop and web applications (consumers) by enabling the consumer to act on behalf of a user without requiring the user to provide sensitive account information, such as username and password.

An example of a consumer might be a company that wants to aggregate all of its customers' photos onto a single web page. Rather than asking every user to enter the username/password for each of their photo sharing accounts, the company can act as a consumer and each of the photo sharing web sites can act as service providers and exchange OAuth tokens. This gives the photo aggregator company access to a user's photos without the user potentially compromising the security of their account by giving out username/password information.

For an individual or organization to become an OAuth service provider, they need to configure their system to support OAuth token requests. There are several ways to do this, but after a number of challenging weeks on a project trying to figure out the best way to integrate OAuth with particular application requirements that included IBM WebSphere Application Server, LTPA, and WS-Security, it became apparent that using Trust Association Interceptors provided the best solution among all the potential approaches that were investigated.

Custom **Trust Association Interceptors** (TAI) enable the integration of vendor-acquired security services with WebSphere Application Server. In this case, OAuth is the security service being used. At a high level, this is how TAIs work:

1. When a request comes into WebSphere Application Server, it passes along the HttpServletRequest and HttpServletResponse to each of the TAIs configured in the WebSphere Application Server security settings.
2. Each individual TAI then determines if it is the intended TAI by looking at the parameters passed in the HttpServletRequest object for the security (authentication or authorization) attributes it has for handling the identity assertion from the security service.
3. The next stage of the TAI then generates the identity using the data supplied in the request and asserts this identity to WebSphere Application Server, which then looks up the user in the configured registry to generate the SSO token.
4. The OAuth custom TAI will check for the required OAuth request parameters and validate the OAuth access token to determine the user to be authenticated and the validity of the token.

In contrast, Table 1 lists problems that were encountered with alternate solutions.

Table 1. Alternatives to Trust Association Interceptors

Alternative	Issues found
JAAS custom login modules	To reconstruct the OAuth tokens provided by the consumer we needed to have access to the HTTP request and response. Unfortunately, this is only available with login modules if the module is part of the WEB_INBOUND JAAS login. Another reason this concept didn't work is

	because we only wanted to challenge basic username/password authentication if the OAuth request parameters were not present, but there is no way to suppress a challenge with a JAAS login module approach. With OAuth requests there is no need to provide the username/password because the access token handles authenticating users.
Custom token generator/consumer	Similar to the custom login modules, we were not able to access the request/response to reconstruct our OAuth tokens. This resulted in an overly complicated approach.
Filters	This approach requires application-specific components. If we wanted to protect resources in multiple web projects we would need to create a filter for each project. Also, using filters would require redeployment with updates.

If you are currently (or want to become) an OAuth service provider, and you have deployed enterprise applications using IBM WebSphere Application Server, then this article will help you setup a custom TAI in order to process OAuth access tokens as an OAuth service provider.

Assumptions

This article assumes a basic understanding of WebSphere Application Server security and the WebSphere Application Server administrative console. Familiarity with the OAuth protocol is recommended.

Also, this article assumes you have service provider code already in place and that you want to enable the use of OAuth tokens to authenticate against your WebSphere Application Server protected services. Details of OAuth service provider code is beyond the scope of this article. See Resources for details on the OAuth protocol and TAIs.

Creating a Custom Trust Association Interceptor

To create a Custom Trust Association, you create a new class that implements **com.ibm.wsspi.security.tai.TrustAssociationInterceptor**. Put this class in its own Java™ project so that it can be exported as a JAR file, independent of any other projects. Once your class implements the TrustAssociationInterceptor, you can add the unimplemented methods (see Resources).

The sample code shown in this article is can be downloaded as a JAR file for use as a reference and for further exploration.

Since you're validating an OAuth token sent from a consumer, you must first verify that the request includes the required OAuth parameters that you're expecting (Listing 1). If the expected request parameters are present, then return `true`, which tells the TAI that it's possible the request is an OAuth request. There is no need to perform complete access token validation at this step. The access token recreation and validation is handled by another method (Listing 2).

Listing 1. Override isTargetInterceptor()

```
@Override
public boolean isTargetInterceptor(HttpServletRequest req)
                throws WebTrustAssociationException {

        boolean target = true;

        Map<String,String> hdrMap = new HashMap<String,String>();
        for (Enumeration<String> hdrs = req.getHeaderNames(); hdrs.hasMoreElements(); ){
                String hd = hdrs.nextElement();
                hdrMap.put(hd,req.getHeader( hd ));
        }
        target &= params.containsKey("oauth_token");
        target &= params.containsKey("oauth_consumer_key");
        target &= params.containsKey("oauth_signature");

        System.out.println("isTargetInterceptor: " + target);

        return target;
}
```

If a request comes into WebSphere Application Server with the required OAuth parameters, then the **negotiateValidateandEstablishTrust** method will be triggered. This is where you should place the logic to validate the OAuth access token from the request parameters (Listing 2). The OAuth service provider implementation might encode other user information into the token and extract this information from the token to be used by the code for making authentication or authorization decisions. If the OAuth service provider logic determines that the resulting accessToken is not valid, then this method must return an unauthorized response. Otherwise, if the OAuth access token is valid, then the TAIResult returned indicates success for the user, as determined through the OAuth access token.

Listing 2. Override negotiateValidateandEstablishTrust()

```
@Override
public TAIResult negotiateValidateandEstablishTrust(
                HttpServletRequest req, HttpServletResponse res)
                throws WebTrustAssociationFailedException {

        Map<String,String> hdrMap = new HashMap<String,String>();
        for (Enumeration<String> hdrs = req.getHeaderNames(); hdrs.hasMoreElements(); ){
                String hd = hdrs.nextElement();
                hdrMap.put(hd,req.getHeader( hd ));
        }

        Object accessToken = null;
        //Below method is specific to our OAuth SP implementation.  Replace:
```

```
        //AccessToken accessToken = OAuthServiceProvider
        //         .GrantResourceAccess(params, request);
        if (accessToken == null) {
                System.out.println("OAuth --- Access token couldn't be created!");
                return TAIResult.create(HttpServletResponse.SC_UNAUTHORIZED);
        }

        //Strip out LTPA token from response here (see listing 3)

        String username = accessToken.getUser();
        return TAIResult.create(HttpServletResponse.SC_OK, username);
}
```

Removing the LTPA token from the response

Many service provider implementations will want to strip out the LTPA token from the response for additional security (Listing 3) because this technology provides single-sign on (SSO) capability. If you do not remove the LTPA token information, the code becomes vulnerable, as the LTPA token granted *will allow whoever has the LTPA token access to all areas of the site that is protected using LTPA*. By stripping it out of the response, OAuth can be accessed as designed.

Listing 3. Strip out the LTPA token (both LTPA version 1 and version 2 cookies

```


        Cookie[] cookies = request.getCookies();
        if(cookies != null){
                for (int i = 0; i < cookies.length; i++) {
                        if (cookies[i].getName().startsWith("LtpaToken")) {
                                cookies[i].setMaxAge(0);
                                cookies[i].setPath("/");
                                response.addCookie(cookies[i]);
                        }
                }
        }
```

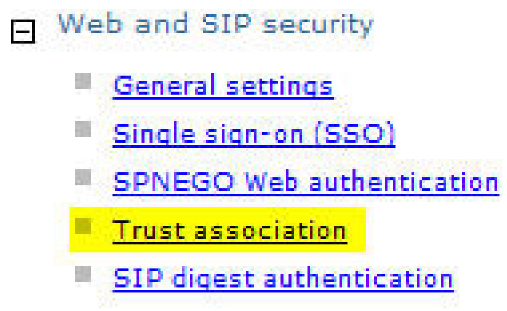
Configuring the TAI in WebSphere Application Server

Once created, the custom TAI can be configured using WebSphere Application Server, following the steps below. (The screen captures shown are from WebSphere Application Server V7.0, but the same process applies V8.0 as well.)

1. Export the TAI project as a JAR file.
2. Place it in the `{$installroot}/lib/ext` directory; for example, `/opt/IBM/WebSphere/AppServer/lib/ext`. (For a clustered environment, make sure this file is available in the `{$installroot}/lib` directory.)
3. Be sure you also add any referenced JAR files to the same directory. This

is important if your OAuth service provider token creation logic is stored in a shared JAR file.

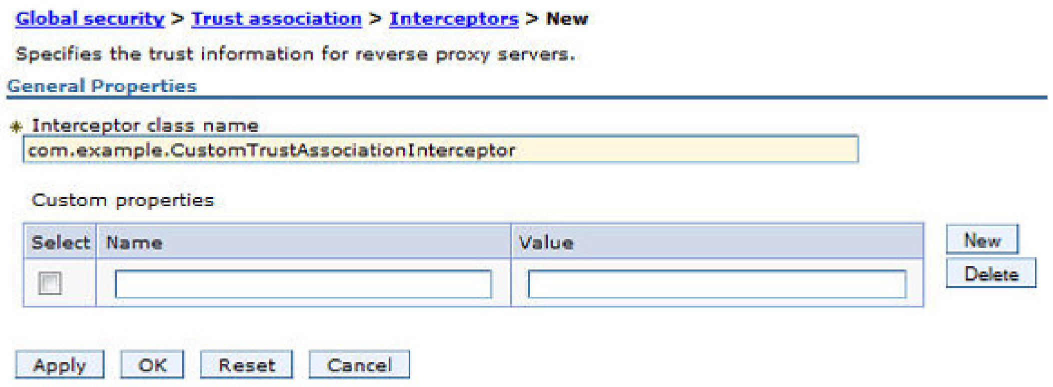
4. Login to the administrative console. Navigate to **Security > Global Security**, and expand **Web and SIP security**. Click on **Trust association** (Figure 1).
Figure 1. Admin console



5. Check **Enable trust association**.

6. Under Additional Properties click **Interceptors**, then click the **New** button.

7. Enter the Interceptor class name (including package information) for your custom TAI (Figure 2). For example: `com.example.CustomTrustAssociationInterceptor`.
Figure 2. Admin console new TAI



Global security > Trust association > Interceptors > New

Specifies the trust information for reverse proxy servers.

General Properties

* Interceptor class name: com.example.CustomTrustAssociationInterceptor

Custom properties

Select	Name	Value
☐		

8. Click **OK**.

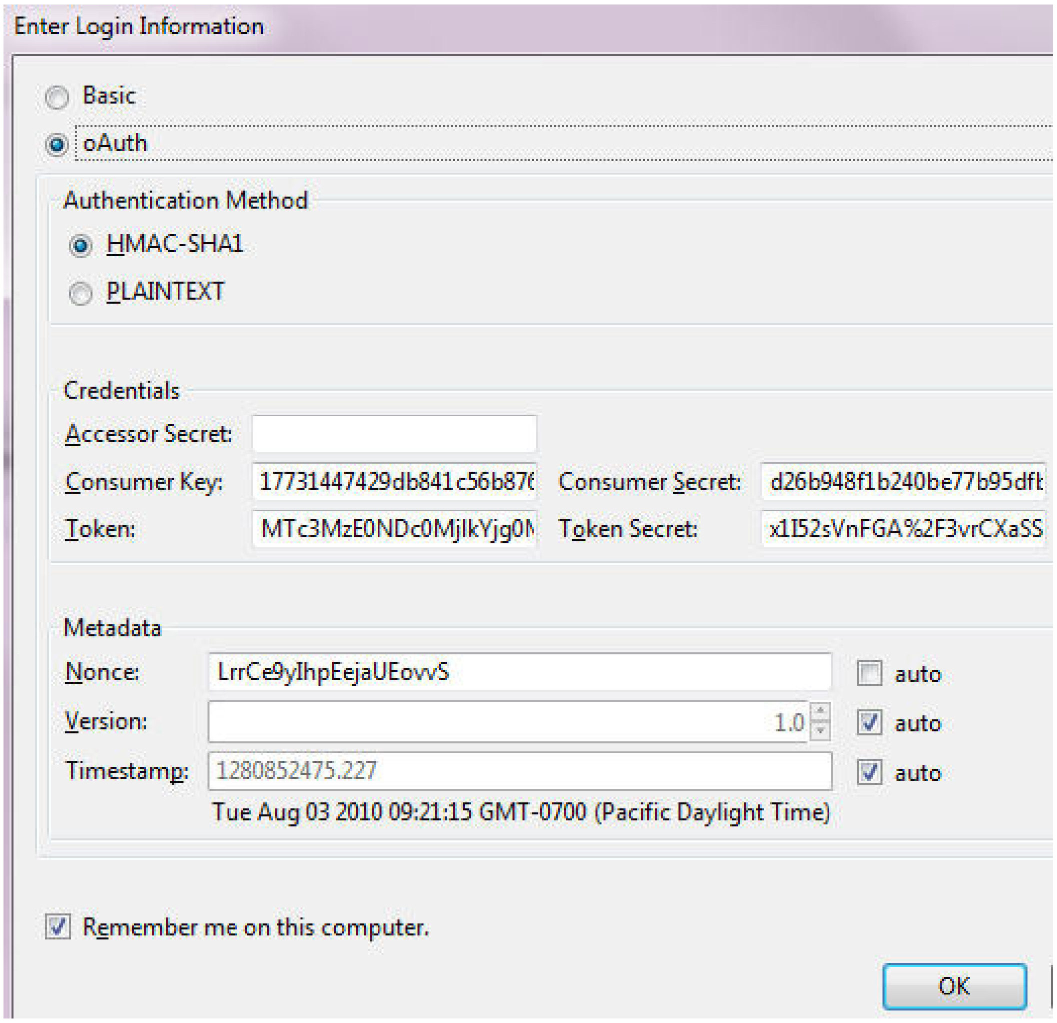
9. Save changes to the master configuration and then restart your server, node, and dmgr to apply the changes you just made.

Testing OAuth

When the restart is complete, you're ready to test to see if the TAI has been configured properly. If you have a consumer OAuth project set up, provide the normal service provider URLs and consumer tokens so you can test OAuth. If you prefer, you can also test the TAI and OAuth with a browser, such as Firefox:

1. Open a Firefox browser and select **Tools > RESTClient**. (See Resources to download the RESTClient Firefox add-on if you do not have it.)
2. On the Enter Login Information panel (Figure 3), select **Login**, then **OAuth**.
3. Check **oAuth** and the appropriate Authentication Method.
4. Enter Credentials and Metadata information associated with your service provider, then click **OK**.

Figure 3. Firefox RESTClient



Enter Login Information

Basic [✓ oAuth]

Authentication Method

[✓ HMAC-SHA1] PLAINTEXT

Credentials

Accessor Secret:			
Consumer Key:	17731447429db841c56b87€	Consumer Secret:	d26b948f1b240be77b95dfb
Token:	MTc3MzE0NDc0MjlkYjg0N	Token Secret:	x1I52sVnFGA%2F3vrCXaSS

Metadata

Nonce:	LrrCe9yIhpEejaUEovvS	☐ auto
Version:	1.0	✓ auto
Timestamp:	1280852475.227	✓ auto

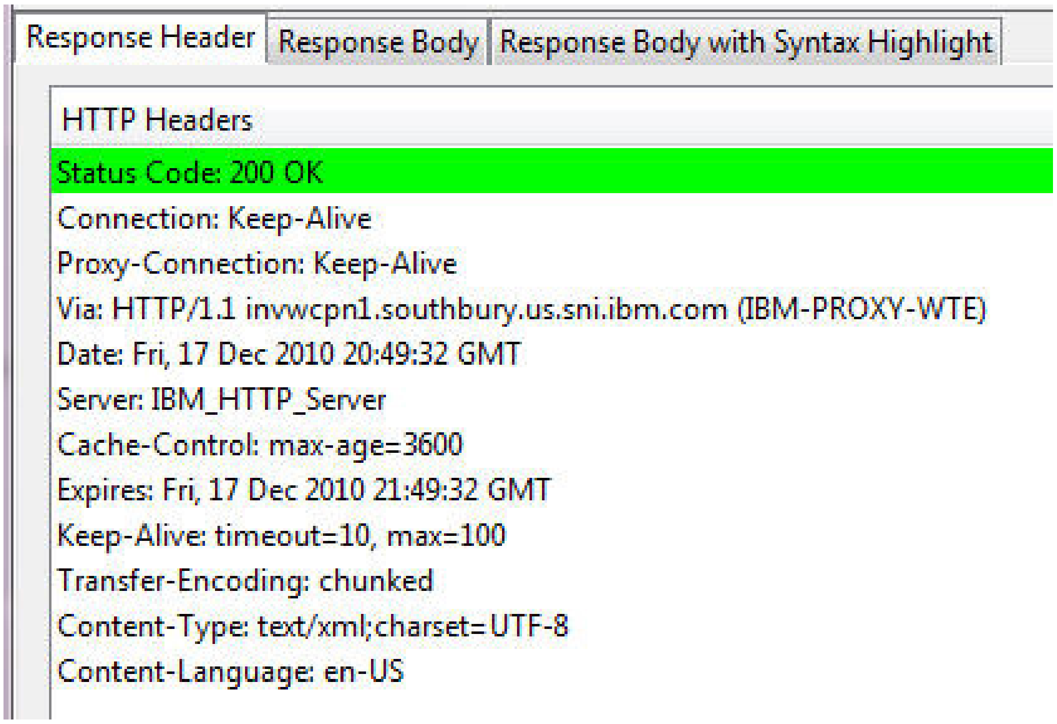
Tue Aug 03 2010 09:21:15 GMT-0700 (Pacific Daylight Time)

✓ Remember me on this computer.

5. Enter the Method and URL for a protected resource provided by the

service provider, the click **Send**. If successful, you will get an appropriate response and an OK status (Figure 4).

Figure 4. Firefox RESTClient



```
Status Code: 200 OK
Connection: Keep-Alive
Proxy-Connection: Keep-Alive
Via: HTTP/1.1 invwcpn1.southbury.us.sni.ibm.com (IBM-PROXY-WTE)
Date: Fri, 17 Dec 2010 20:49:32 GMT
Server: IBM_HTTP_Server
Cache-Control: max-age=3600
Expires: Fri, 17 Dec 2010 21:49:32 GMT
Keep-Alive: timeout=10, max=100
Transfer-Encoding: chunked
Content-Type: text/xml;charset=UTF-8
Content-Language: en-US
```

Next steps

Now that the WebSphere Application Server TAI is up and running, consumers can successfully connect via OAuth to your OAuth service provider, gaining secure access to protected resources. Take a closer look at your service provider implementation to ensure that the set of protected resources are intended and that any other security concerns are handled, then adjust the token granting logic accordingly. See Resources to learn more on the OAuth protocol and TAIs.

Conclusion

This information should help you get started with using OAuth tokens with WebSphere Application Server. Using a TAI turned out to be not only the easiest solution to implement, but the most logical for the deployment needs of our project. However, it's critical to keep in mind that when using a TAI, any OAuth request could potentially permit consumer access to every resource that is actively being protected by other security methods, such as WS-Security and LTPA tokens. Your OAuth service provider code is responsible for determining which specific resource the consumer is attempting to access, and restricting that access based on the URL.

This logic should reside with the service provider's OAuth token granting implementation. Stripping out the LTPA token, as described earlier, should also be done to limit access to only legitimate requests.

Downloads

Description	Name	Size	Download method
Sample code	customoAuthTAI.jar	4KB	HTTP

Information about download methods

Resources

Learn

- OAuth specification and documentation
- OAuth Core 1.0 Revision A specification
- IBM developerWorks Open source zone
- Information Center: Trust Association Interceptor
- Advanced authentication in WebSphere Application Server
- IBM developerWorks WebSphere

Get products and technologies

- Provision cloud resources: IBM SmartCloud Innovation Center
- Firefox Add-Ons: RESTClient

Discuss

- Firefox Add-Ons: Become a Partner and Leverage the Power of the IBM Smart Business Development & Test API

About the authors

Lisa Seacat DeLuca
Lisa Seacat DeLuca is an Advisory Software Engineer for the Advanced Cloud Technologies team under IBM's Global Technology Services, GTS. Lisa has over six years of Enterprise Software Development experience and led the OAuth support effort for the IBM SmartCloud offering. She holds a Masters of Science in Technology Commercialization from the University of Texas and a Bachelors of Science in Computer Science from Carnegie Mellon University.

Albert A DeLucca
Al DeLucca is a Senior Software Engineer working on Advanced Cloud Technologies in IBM's Global Technology Services, where his primary role is architecture and design of IBM's cloud offerings, development of cloud standards, and building innovative cloud solutions. He has over 8 years of experience in software design, service oriented architectures, and highly available computing. His recent work involves developing new cloud consumption models, flexible web

frameworks, and the design and delivery of IBM's SmartCloud Innovation Center.

Ryan DeJana
Ryan DeJana is an IBM Certified IT Specialist and Software Engineer based in Boulder, Colorado. He is currently working in IBM's Advanced Cloud Solutions and Innovation organization, where he is responsible for architecture, development and end-to-end user experience for IBM's public cloud offerings. He has over a decade of experience in leading the design and development of enterprise applications, from mainframes to highly available web portals. His recent work involves securely and efficiently moving data in and out of clouds and in the design and delivery of IBM's SmartCloud Innovation Center.

JVM updates in WebSphere Application Server V8:

Using wsadmin and Jython to easily collect and report WebSphere Application Server PMI data

Skill Level: Intermediate

Denis Guillemenot (denis_guillemenot@fr.ibm.com)
Senior IT Consultant
IBM

07 Dec 2011

This article describes how you can use wsadmin introspection with Jython to easily retrieve Performance Monitoring Infrastructure (PMI) metrics for IBM® WebSphere® Application Server. IBM Tivoli® Performance Viewer is used as a model for performance reporting. A high level review of PMI architecture is included.

Introduction

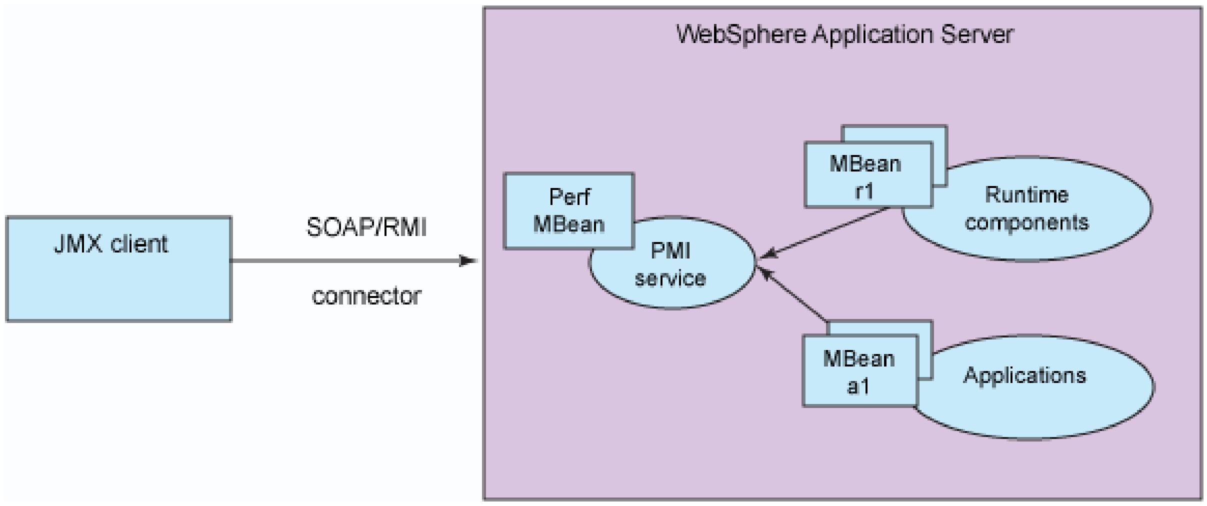
IBM WebSphere Application Server provides performance metrics through the Performance Monitoring Infrastructure (PMI). WebSphere Application Server uses PMI to collect performance (and statistical) metrics data when running. The number and scope of metrics collected depends on the level of information needed.

There are several important things you need to know and remember about WebSphere Application Server and PMI. On each WebSphere Application Server JVM, the PMI framework is composed of a:

- **PMI service**, which must be activated to collect data (on NodeAgent also).
- **PMI module**, which permits you to configure what data to collect.
- **Performance Mbean** (PefMBean), which permits the configuration of the PMI service and the gathering of data.

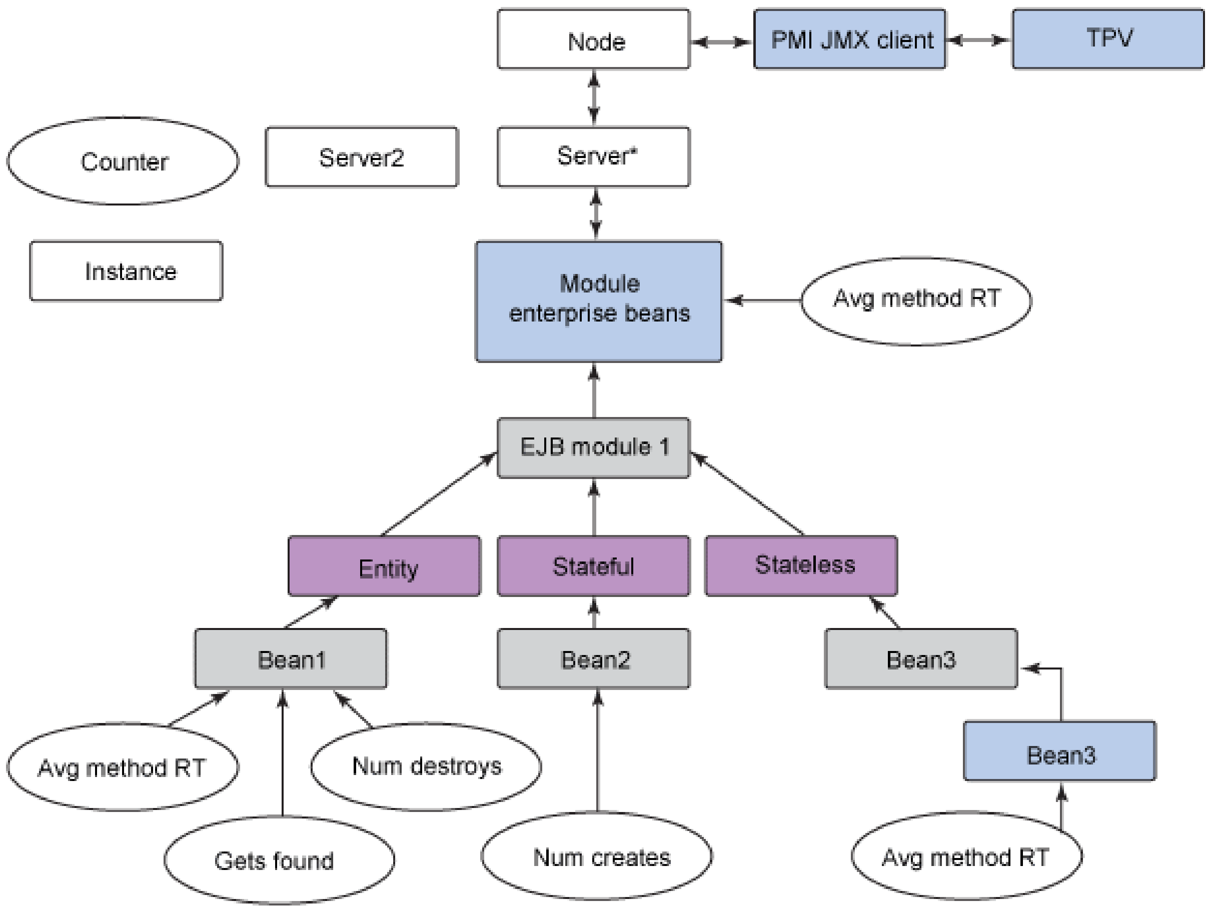
Because PMI conforms to the Java™ Management eXtension (JMX) standard, PMI data is accessible via the JMX interface (Figure 1).

Figure 1. JMX interface



JMX is a management model where resources are managed by managed beans (Mbeans). Some Mbeans are statistics providers and have a **Stats** attribute. A PerfMbean can collect data from multiple Mbeans in one JMX call. For each application server, PMI data is collected and stored in a hierarchical tree-like structure with modules, submodules, and counters (Figure 2); if you ask the PMI data for the server Mbean, then you have all the PMI data of the server.

Figure 2. Hierarchy of data collections used for performance reporting to Tivoli Performance Viewer



Get more information

See the WebSphere Application Server Information Center documentation to get a better understanding of PMI data organization.

To access the final metrics, you need to select the applicable modules, submodules, and counters on which you want to report. IBM Tivoli Performance Viewer is an excellent and popular graphical tool to help you do this. Taking this a step further, it would be great if you could browse modules and access metrics using wsadmin or Jython as easily as you can with Tivoli Performance Viewer. The good news is that you can!

To show how you can do this, we'll start with the techniques described in an earlier article, Writing PMI applications using the JMX interface. The examples in that article were created for Java developers. Here, we'll use wsadmin Jython scripts to demonstrate how easy it can be using this method to read and understand performance metrics.

However, before you can retrieve PMI metrics, you first need to understand how to use introspection and how to call MBean JMX methods (via the PerfMBean). These topics are discussed in the next sections.

The power of introspection

In general, **introspection** refers to the ability to examine an element, component, or object to determine what it is, what it knows, or what it can do. Introspection plays an important part in successfully implementing both wsadmin and Jython.

wsadmin introspection

The wsadmin utility includes five scripting objects:

- **AdminControl** to manage running objects.
- **AdminConfig** to manage configuration objects.
- **AdminApp** to manage application objects.
- **AdminTask** to make repetitive or complex tasks easier.
- **Help** to provide help about scripting objects.

To launch wsadmin using Jython, just type the command: `<WAS_PATH>/wsadmin.sh -lang jython`.

You can use the wsadmin Help object to display information about each object using these commands:

- `print Help()`
- `print Help.help()`
- `print Help.AdminControl()`
- `print Help.AdminConfig()`
- `print Help.AdminApp()`
- `print Help.AdminTask()`
- `print Help.operations( <object name string>)`
- `print Help.attributes( <object name string>)`

You can get almost all the known object types with by typing `AdminConfig.types()`; that's “almost” because all the possible types used in `AdminControl.queryNames('type=...')` are not listed with AdminConfig.types().

Listing 1 shows an example of **Help.attributes()** and **Help.operations()**, where all Mbeans of type **JVM** are listed and put in an Jython list, and then all attributes and methods of the JVM server1 are listed (`jvmList[2]`). The `stats` attribute and the **getStats()** method are important for PMI data.

Listing 1. wsadmin introspection: Help.attributes() and Help.operations()

```
$ /websphere/mywas/bin/wsadmin.sh -lang jython
WASX7209I: Connected to process "dmgr" on node Node01 using SOAP connector;  The type of
process is: DeploymentManager
WASX7031I: For help, enter: "print Help.help()"
wsadmin>

wsadmin> jvmList = AdminControl.queryNames( 'type=JVM,*').split( lineSeparator)
['WebSphere:name=JVM,process=dmgr,platform=proxy,node=W2K300Dmgr,j2eeType=JVM,J2EEServer=
dmgr,version=7.0.0.5,type=JVM,mbeanIdentifier=JVM,cell=W2K300Cell,spec=1.0', 'WebSphere:
name=JVM,process=nodeagent,platform=proxy,node=W2K300Node01,j2eeType=JVM,J2EEServer=
nodeagent,version=7.0.0.5,type=JVM,mbeanIdentifier=JVM,cell=W2K300Cell,spec=1.0',
'WebSphere:name=JVM,process=server1,platform=proxy,node=W2K300Node01,j2eeType=JVM,
J2EEServer=server1,version=7.0.0.5,type=JVM,mbeanIdentifier=JVM,cell=W2K300Cell,spec=1.0',
'WebSphere:name=JVM,process=server2,platform=proxy,node=W2K300Node01,j2eeType=JVM,
J2EEServer=server2,version=7.0.0.5,type=JVM,mbeanIdentifier=JVM,cell=W2K300Cell,spec=1.0']

wsadmin> print Help.attributes( jvmList[2])
Attribute                       Type                            Access
javaVendor                      java.lang.String                RO
javaVersion                     java.lang.String                RO
node                            java.lang.String                RO
heapSize                        java.lang.String                RO
freeMemory                      java.lang.String                RO
maxHeapDumpsOnDisk              java.lang.Integer               RW
maxMemory                       java.lang.String                RO
maxSystemDumpsOnDisk            java.lang.Integer               RW
stats                           javax.management.j2ee.statistics.Stats  RO
objectName                      java.lang.String                RO
stateManageable                 boolean                         RO
statisticsProvider              boolean                         RO
eventProvider                   boolean                         RO

wsadmin> print Help.operations( jvmList[2])
Operation
java.lang.String getJavaVendor()
java.lang.String getJavaVersion()
java.lang.String getJVMNode()
java.lang.String getTotalMemory()
java.lang.String getFreeMemory()
java.lang.Integer getMaxDumpsOnDisk()
void setMaxDumpsOnDisk(java.lang.Integer)
java.lang.String getMaxMemory()
java.lang.Integer getMaxSystemDumpsOnDisk()
void setMaxSystemDumpsOnDisk(java.lang.Integer)
void dumpThreads()
java.lang.String getProperty(java.lang.String)
java.lang.String getIPAddress(java.lang.String)
long getCurrentTimeInMillis()
java.lang.String generateSystemDump()
java.lang.String; generateSystemDump(java.lang.String)
java.lang.String generateHeapDump()
[Ljava.lang.String; generateHeapDumps()
[Ljava.lang.String; generateHeapDump(java.lang.String)
boolean isVerbose()
void setVerbose(boolean, java.lang.String)
javax.management.j2ee.statistics.Stats getStats()
java.lang.String getObjectNameStr()
boolean isStateManageable()
boolean isStatisticsProvider()
boolean isEventProvider()
```

```
wsadmin> jvmStr = AdminControl.queryNames( 'type=JVM,process=server1,*')
wsadmin> jvmStr
'WebSphere:name=JVM,process=server1,platform=proxy,node=W2K300Node01,j2eeType=JVM,
J2EEServer=server1,version=7.0.0.5,type=JVM,mbeanIdentifier=JVM,cell=W2K300Cell,spec=1.0'
```

Listing 2 shows an example of **AdminConfig.types()**.

Listing 2. wsadmin introspection: AdminConfig.types()

```
wsadmin> print AdminConfig.types()
AccessPointGroup
ActivationSpec
ActivationSpecTemplateProps
ActivitySessionService
…
WorkManagerInfo
WorkManagerProvider
WorkManagerService
WorkloadManagementServer

wsadmin> [p for p in AdminConfig.types().split() if p.lower().find('pmi')>=0]
['PMIModule', 'PMIRMFilter', 'PMIRMFilterValue', 'PMIRequestMetrics', 'PMIService']
```

The Python syntax `[var for var in …]` is called **list comprehension** (see the Python tutorial in Resources).

Jython and Java introspection

As it is in Python, introspection is key when programming in Jython. Because Jython manipulates Python objects and Java objects, there are two ways to perform introspection:

- **Python objects:** Python introspection with **type()** and **dir()**.
- **Java objects:** Java introspection with **getClass()** and **getMethods()**.

Be sure to remember that all Java objects are created from the **class java.lang.Object** and **java.lang.Class** (see Resources) so you get access to all their methods from Jython.

Now, let's verify. In Listing 3, you launch wsadmin with Jython and then use **dir()** and **type()** to look at the available objects:

- **Get known objects:** AdminApp, AdminConfig, … sys, …
- **Query an object:** dir(AdminApp), dir(sys.version)
- **Get the type of an object:** type(AdminApp), type(sys.version)

Listing 3. Jython introspection: dir() and type()

```
wsadmin> dir()
['AdminApp', 'AdminConfig', 'AdminControl', 'AdminTask', 'Help', 'LTPA_LDAPSecurityOff',
'LTPA_LDAPSecurityOn', 'TypedProxy', '__doc__', '__name__', 'bsf', 'cellName',
'checkuserpw', 'doAuthenticationMechanism', 'doGlobalSecurity',
'doGlobalSecurityDisable', 'doLDAPUserRegistry', 'domainHostname', 'exportLTPAKey',
'flag', 'forceSync', 'generateLTPAKeys', 'getLDAPUserRegistryId', 'getLTPAId',
'getSecId', 'getSecurityAdminMbean', 'java', 'ldapPassword', 'ldapPort', 'ldapServer',
'ldapServerId', 'ldapUserRegistryId', 'lineSeparator', 'ltpaId', 'nodeName',
'secMbean', 'securityId', 'securityoff', 'securityon', 'sleep', 'sys', 'whatEnv']

wsadmin> dir( AdminApp)
[]

wsadmin> type( AdminApp)
<jclass org.python.core.PyJavaInstance at 522592038>

wsadmin> dir( sys)
['__delattr__', '__dict__', '__displayhook__', '__excepthook__', '__findattr__',
'__setattr__', '__stderr__', '__stdin__', '__stdout__', '_getframe', 'add_classdir',
'add_extdir', 'add_package', 'argv', 'builtin_module_names', 'builtins', 'cachedir',
'classLoader', 'copyright', 'defaultencoding', 'displayhook', 'exc_info', 'excepthook',
'exec_prefix', 'executable', 'exit', 'getClassLoader', 'getdefaultencoding',
'hexversion', 'initialize', 'last_traceback', 'last_type', 'last_value', 'maxint',
'minint', 'modules', 'packageManager', 'path', 'platform', 'prefix', 'profile',
'profilefunc', 'ps1', 'ps2', 'registry', 'setClassLoader', 'setdefaultencoding',
'setprofile', 'settrace', 'stderr', 'stdin', 'stdout', 'toString', 'trace',
'tracefunc', 'version', 'version_info', 'warnoptions']

wsadmin> print sys.version
2.1

wsadmin> dir( sys.version)
[]

wsadmin> type( sys.version)
<jclass org.python.core.PyString at 469769216>
```

In Listing 4, **getClass()** and **getMethods()** are used to look at the JVM Mbean methods:

- **Search object string name:** AdminControl.queryNames()
- **Get object reference:** AdminControl.makeObjectName()
- **Query an object:** <object>.getClass(), <object>.getMethods()

Be aware that getClass() and getMethods() only work on Jython objects of type **org.python.core.PyJavaInstance** and not on Jython objects of type **org.python.core.PyString**.

Listing 4. Java introspection: getClass() and getMethods()

```


wsadmin> type( jvmList)
<jclass org.python.core.PyList at 778972782>

wsadmin> type( jvmStr)
<jclass org.python.core.PyString at 1555061936>

wsadmin> jvmStr.getClass()
WASX7015E: Exception running command: "jvmList[2].getClass()"; exception information:
 com.ibm.bsf.BSFException: exception from Jython:
Traceback (innermost last):
  File "<input>", line 1, in ?
AttributeError: 'string' object has no attribute 'getClass'

wsadmin> jvmObj = AdminControl.makeObjectName( jvmStr)
wsadmin> jvmObj
WebSphere:name=JVM,process=server1,platform=proxy,node=W2K300Node01,j2eeType=JVM,
J2EEServer=server1,version=7.0.0.5,type=JVM,mbeanIdentifier=JVM,cell=W2K300Cell,spec=1.0

wsadmin> type( jvmObj)
<jclass org.python.core.PyJavaInstance at 2093776076>

wsadmin> jvmObj.getClass()
<jclass javax.management.ObjectName at 1904111998>

wsadmin> type( jvmObj.getClass())
<jclass org.python.core.PyJavaClass at 1378767406>

wsadmin> type( jvmObj.getClass().getMethods())
<jclass org.python.core.PyArray at 90441060>

wsadmin> for i in jvmObj.getClass().getMethods(): print i
public boolean javax.management.ObjectName.equals(java.lang.Object)
public int javax.management.ObjectName.hashCode()
public java.lang.String javax.management.ObjectName.toString()
public boolean javax.management.ObjectName.isPattern()
public boolean javax.management.ObjectName.isDomainPattern()
public boolean javax.management.ObjectName.isPropertyPattern()
public boolean javax.management.ObjectName.isPropertyListPattern()
public boolean javax.management.ObjectName.isPropertyValuePattern()
public boolean javax.management.ObjectName.isPropertyValuePattern(java.lang.String)
throws java.lang.NullPointerException,java.lang.IllegalArgumentException
public java.lang.String javax.management.ObjectName.getCanonicalName()
public java.lang.String javax.management.ObjectName.getDomain()
public java.lang.String javax.management.ObjectName.getKeyProperty(java.lang.String)
throws java.lang.NullPointerException
public java.util.Hashtable javax.management.ObjectName.getKeyPropertyList()
public java.lang.String javax.management.ObjectName.getKeyPropertyListString()
...
public final native java.lang.Class java.lang.Object.getClass()
...
```

Be careful: jvmObj is of type **org.python.core.PyJavaInstance**; that is, the Jython implementation of a Java class. The getClass() method gives the real Java class name **javax.management.ObjectName**.

Object name

Object name refers to an active JMX Mbean that can be controlled with one of the commands in the AdminControl scripting object. The syntax is the standard JMX object name syntax:

```
domain:key=value,key=value,key=value,...
```

For WebSphere Application Server, the domain is WebSphere:

```
WebSphere:key=value,key=value,key=value,...
```

To get an object name, you can use:

- **AdminControl.queryNames(<string>)** returns a string
- **AdminControl.makeObjectName(<string>)** returns an object name

Therefore, jvmObj is a Java object of type **javax.management.ObjectName**, which points to the desired Mbean. To get the methods of the Mbean, you cannot use the getMethods() method directly on the jvmObj object. Instead, you need to use Help.operations(), as shown in Listing 1.

Calling MBean JMX methods via the PerfMBean

To simplify, there are three kind of PMI related Mbeans:

- **PerfMBean** to manage and control other MBeans
- **StatisticsProvider** Mbeans for providing the stats attribute containing statistics.
- **Other** MBeans without stats attribute.

The power of introspection enables you to retrieve all the information you need to call Mbeans methods by:

- Listing the methods of a Mbean.
- Figuring the parameters needed.
- Constructing the signature.

As you can see in Listing 1, the JVM Mbean has a stats attribute and a getStats() method, but you cannot call this method directly (Listing 5).

Listing 5. Trying to get stats object directly

```
wsadmin> jvmObj.getStats()
WASX7015E: Exception running command: "jvmObj.getStats()"; exception information:
 com.ibm.bsf.BSFException: exception from Jython:
Traceback (innermost last):
  File "<input>", line 1, in ?
AttributeError: getStats
```

To execute Mbean methods, you must use one of these techniques:

- **AdminControl.invoke(<mbeanStr>, 'method', <parameters>)**, where 'method' is a call to a method of the <mbeanStr>.
- **AdminControl.invoke_jmx(<perfObj>, 'method', <parameters>,**

<signature>), where 'method' is a call to a method of the <perfObj> PerfMBean and <parameters> will contain the name of the target <mbeanObj>.

To use the AdminControl.invoke() method, you need to provide the string version of the Mbean and you get the string version of the stats object (Listing 6).

Listing 6. Getting the string stats object with AdminControl.invoke()

```
wsadmin>print statStr = AdminControl.invoke( jvmStr, 'getStats')
Stats name=jvmRuntimeModule, type=jvmRuntimeModule#
{
name=HeapSize, ID=1, description=The total memory (in KBytes) in the Java virtual
machine run time., unit=KILOBYTE, type=BoundedRangeStatistic, lowWaterMark=51200,
highWaterMark=70012, current=70012, integral=0.0, lowerBound=51200, upperBound=262144

name=UsedMemory, ID=3, description=The amount of used memory (in KBytes) in the Java
virtual machine run time., unit=KILOBYTE, type=CountStatistic, count=69008

name=UpTime, ID=4, description=The amount of time (in seconds) that the Java virtual
machine has been running., unit=SECOND, type=CountStatistic, count=77341

name=ProcessCpuUsage, ID=5, description=The CPU Usage (in percent) of the Java virtual
machine., unit=N/A, type=CountStatistic, count=0
}

wsadmin>type( statStr)
<jclass org.python.core.PyString at 1555061936>
```

To use the AdminControl.invoke_jmx() method, you need to provide the object version of the MBean (type **javax.management.ObjectName**) and you will get the object version of the stats object. But for this, you need to use the PerfMBean associated with this JVM, and you'll use the getKeyPropertyList() method to get it (Listing 7).

Listing 7. Getting the PerfMBean of the server

```
wsadmin> jvmObj.getKeyPropertyList()
{version=7.0.0.5, j2eeType=JVM, platform=proxy, mbeanIdentifier=JVM, process=server1,
type=JVM, name=JVM, node=W2K300Node01, spec=1.0, J2EEServer=server1, cell=W2K300Cell}

wsadmin> jvmObj.getKeyPropertyList()['process']
'server1'

wsadmin> perfStr = AdminControl.queryNames('type=Perf,process=%s,*' %
jvmObj.getKeyPropertyList()['process'])
'WebSphere:name=PerfMBean,process=server1,platform=dynamicproxy,node=W2K300Node01,
version=7.0.0.5,type=Perf,mbeanIdentifier=PerfMBean,cell=W2K300Cell,spec=1.0'

wsadmin>type( perfStr)
<jclass org.python.core.PyString at 1555061936>

wsadmin>perfObj = AdminControl.makeObjectName( perfStr)
wsadmin>perfObj
WebSphere:name=PerfMBean,process=server1,platform=dynamicproxy,node=W2K300Node01,
version=7.0.0.5,type=Perf,mbeanIdentifier=PerfMBean,cell=W2K300Cell,spec=1.0

wsadmin>type( perfObj)
```

```
<jclass org.python.core.PyJavaInstance at 1467504504>
```

You can now list all the methods available with the PerfMBean (Listing 6).

Listing 8. Getting PerfMBean methods

```
wsadmin>print Help.attributes( perfStr)
Attribute                       Type                            Access

wsadmin>print Help.operations( perfStr)
Operation
com.ibm.websphere.pmi.PmiModuleConfig getConfig(javax.management.ObjectName)
com.ibm.websphere.pmi.PmiModuleConfig getConfig(java.lang.String)
[Lcom.ibm.websphere.pmi.PmiModuleConfig; getConfigs(java.util.Locale)
com.ibm.websphere.pmi.stat.WSStats getStatsObject(javax.management.ObjectName,
java.lang.Boolean)
[Lcom.ibm.websphere.pmi.stat.WSStats; getStatsArray([Ljavax.management.ObjectName;,
java.lang.Boolean)
[Lcom.ibm.websphere.pmi.stat.WSStats; getStatsArray([Lcom.ibm.websphere.pmi.stat.
StatDescriptor;, java.lang.Boolean)
[Lcom.ibm.websphere.pmi.stat.MBeanLevelSpec; getInstrumentationLevel(javax.management.
ObjectName, java.lang.Boolean)
[Lcom.ibm.websphere.pmi.stat.StatLevelSpec; getInstrumentationLevel(com.ibm.websphere.
pmi.stat.StatDescriptor, java.lang.Boolean)
java.lang.String getStatisticSet()
java.lang.String getCustomSetString()
[Lcom.ibm.websphere.pmi.stat.StatDescriptor; listStatMembers(com.ibm.websphere.pmi.stat.
StatDescriptor, java.lang.Boolean)
void setInstrumentationLevel(com.ibm.websphere.pmi.stat.MBeanLevelSpec, java.lang.Boolean)
void setInstrumentationLevel([Lcom.ibm.websphere.pmi.stat.MBeanLevelSpec;,
java.lang.Boolean)
void setInstrumentationLevel([Lcom.ibm.websphere.pmi.stat.StatLevelSpec;,
java.lang.Boolean)
void setStatisticSet(java.lang.String)
void setCustomSetString(java.lang.String, java.lang.Boolean)
void savePMIConfiguration()
[Lcom.ibm.websphere.pmi.PmiModuleConfig; getConfigs()
com.ibm.websphere.pmi.stat.WSStats getStatsObject(com.ibm.websphere.pmi.stat.
MBeanStatDescriptor, java.lang.Boolean)
com.ibm.websphere.pmi.stat.WSStats getStatsObject(com.ibm.ws.pmi.server.DataDescriptor,
java.lang.Boolean)
[Lcom.ibm.websphere.pmi.stat.WSStats; getStatsArray([Lcom.ibm.websphere.pmi.stat.
MBeanStatDescriptor;, java.lang.Boolean)
[Lcom.ibm.websphere.pmi.stat.WSStats; getStatsArray([Lcom.ibm.ws.pmi.server.
DataDescriptor;, java.lang.Boolean)
java.lang.String getStatsString(javax.management.ObjectName, java.lang.Boolean)
  <deprecated>
java.lang.String getStatsString(javax.management.ObjectName, java.lang.String,
java.lang.Boolean)  <deprecated>
void setInstrumentationLevel(java.lang.String, java.lang.Boolean)
void setInstrumentationLevel(com.ibm.ws.pmi.server.PerfLevelDescriptor, java.lang.Boolean)
void setInstrumentationLevel([Lcom.ibm.ws.pmi.server.PerfLevelDescriptor;,
java.lang.Boolean)
[Lcom.ibm.websphere.pmi.stat.MBeanLevelSpec; getInstrumentationLevel
(com.ibm.websphere.pmi.
stat.MBeanStatDescriptor, java.lang.Boolean)
[Lcom.ibm.ws.pmi.server.PerfLevelDescriptor; getInstrumentationLevel
(com.ibm.ws.pmi.server.
DataDescriptor, java.lang.Boolean)
java.lang.String getInstrumentationLevelString()  <deprecated>
[Lcom.ibm.websphere.pmi.stat.MBeanStatDescriptor; listStatMembers(javax.management.
ObjectName)
[Lcom.ibm.websphere.pmi.stat.MBeanStatDescriptor; listStatMembers(com.ibm.websphere.pmi.
stat.MBeanStatDescriptor)
[Lcom.ibm.ws.pmi.server.DataDescriptor; listStatMembers(com.ibm.ws.pmi.server.
DataDescriptor)
java.lang.String listStatMemberNames(javax.management.ObjectName)  <deprecated>
```

Next, you are going to invoke the JMX **getStatsObject()** method from the PerfMBean to get a com.ibm.websphere.pmi.stat.WSStats object, which is the stats attribute of the JVM MBean.

The AdminControl.invoke_jmx() requires four parameters:

1. PerfMbean object of the server managing the target Mbean.
2. The method of PerfMBean to call.
3. The parameters of the called method, obtained with Help.operations().
4. The "signature" of the parameters; that is, the type of each parameter, also obtained with Help.operations().

You might have noticed that there are three versions of the getStatsObject(), with three differentes signatures; you can use whichever is easier for you, but be careful as some are deprecated.

Let's get the stats attribute in object format of the JVM MBean using the non-deprecated getStatsObject() method: **com.ibm.websphere.pmi.stat.WSStats getStatsObject(javax.management.ObjectName, java.lang.Boolean** (Listing 9).

Listing 9. Get the “object” stats attribute of the JVM Mbean

```
wsadmin> params = [ jvmObj, java.lang.Boolean ('true')]

wsadmin> sigs = ['javax.management.ObjectName', 'java.lang.Boolean']

wsadmin> jvmStats = AdminControl.invoke_jmx( perfObj, 'getStatsObject', params, sigs)
Stats name=jvmRuntimeModule, type=jvmRuntimeModule#
{
name=HeapSize, ID=1, description=The total memory (in KBytes) in the Java virtual
machine run time., unit=KILOBYTE, type=BoundedRangeStatistic, lowWaterMark=51200,
highWaterMark=60149, current=60149, integral=0.0, lowerBound=51200, upperBound=262144
name=UsedMemory, ID=3, description=The amount of used memory (in KBytes) in the Java
virtual machine run time., unit=KILOBYTE, type=CountStatistic, count=59102
name=UpTime, ID=4, description=The amount of time (in seconds) that the Java virtual
machine has been running., unit=SECOND, type=CountStatistic, count=19513005
name=ProcessCpuUsage, ID=5, description=The CPU Usage (in percent) of the Java virtual
machine., unit=N/A, type=CountStatistic, count=0
}
```

You can check the type and methods of the stats object as shown in Listing 10.

Listing 10. Analyze the stats object with introspection

```
wsadmin> type (jvmStats)
<jclass org.python.core.PyJavaInstance at 1671979944>

wsadmin>jvmStats.getClass()
<jclass com.ibm.ws.pmi.stat.StatsImpl at 1708942812>

wsadmin> for i in jvmStats.getClass().getMethods(): print i

public boolean java.lang.Object.equals(java.lang.Object)
public native int java.lang.Object.hashCode()
public java.lang.String com.ibm.ws.pmi.stat.StatsImpl.toString()
public java.lang.String com.ibm.ws.pmi.stat.StatsImpl.getName()
public int com.ibm.ws.pmi.stat.StatsImpl.numStatistics()
…
public com.ibm.websphere.pmi.stat.WSStatistic
        com.ibm.ws.pmi.stat.StatsImpl.getStatistic(int)
public com.ibm.websphere.pmi.stat.WSStatistic
        com.ibm.ws.pmi.stat.StatsImpl.getStatistic(java.lang.String)
public com.ibm.websphere.pmi.stat.WSStatistic[]
        com.ibm.ws.pmi.stat.StatsImpl.getStatistics()
public com.ibm.websphere.pmi.stat.WSStatistic[]
        com.ibm.ws.pmi.stat.StatsImpl.listStatistics()
public java.lang.String[] com.ibm.ws.pmi.stat.StatsImpl.listStatisticNames()
public java.lang.String[] com.ibm.ws.pmi.stat.StatsImpl.getStatisticNames()
public com.ibm.ws.pmi.stat.StatisticImpl
        com.ibm.ws.pmi.stat.StatsImpl.getJ2EEStatistic(java.lang.String)
public com.ibm.ws.pmi.stat.StatisticImpl[]
        com.ibm.ws.pmi.stat.StatsImpl.getJ2EEStatistics()
public java.lang.String[] com.ibm.ws.pmi.stat.StatsImpl.getJ2EEStatisticNames()
...
```

You can now retrieve information from the stats object using:

- **numStatistics():** number of PMI metrics available.
- **getJ2EEStatistics():** get all the PMI metrics available.
- **toXML():** get all the PMI metrics available in XML format.

Listing 11. Getting all the stats from the stats object

```


wsadmin> jvmStats.numStatistics()
4

wsadmin> jvmStats.listStatisticNames()
array(['HeapSize', 'UsedMemory', 'UpTime', 'ProcessCpuUsage'], java.lang.String)

wsadmin> print jvmStats.toXML()
<Stats name="jvmRuntimeModule" statType="jvmRuntimeModule#" il="-2" type="COLLECTION">
<BRS id="1" lWM="51200" hWM="60149" cur="60149" int="0.0" sT="1269269566329"
        lST="1288782571946" lB="51200" uB="262144">
</BRS>
<CS id="3" sT="1269269566327" lST="1288867956052" ct="48753">
</CS>
<CS id="4" sT="1269269566327" lST="1288867956052" ct="19598389">
</CS>
<CS id="5" sT="1269269566319" lST="1288867956052" ct="0">
</CS>
</Stats>

wsadmin> jvmStats.getJ2EEStatistics()
array([name=HeapSize, ID=1, description=The total memory (in KBytes) in the Java virtual
machine run time., unit=KILOBYTE, type=BoundedRangeStatistic, lowWaterMark=51200,
highWaterMark=60149, current=60149, integral=0.0, lowerBound=51200, upperBound=262144,
name=UsedMemory, ID=3, description=The amount of used memory (in KBytes) in the Java
virtual machine run time., unit=KILOBYTE, type=CountStatistic, count=48753, name=UpTime,
ID=4, description=The amount of time (in seconds) that the Java virtual machine has been
running., unit=SECOND, type=CountStatistic, count=19598389, name=ProcessCpuUsage, ID=5,
description=The CPU Usage (in percent) of the Java virtual machine., unit=N/A,
type=CountStatistic, count=0], com.ibm.ws.pmi.stat.StatisticImpl)
```

The last method call, **getJ2EEStatistics()**, shows that there are different types of statistics. Indeed, there are five types of statistics objects (listed here with some of their methods):

- **Stats:** getStatistics(), getStatistic(), getStatisticNames(), getSubStats()
- **CountStatistic:** getCount()
- **TimeStatistic:** getCount(), getMean()
- **RangeStatistic:** getCurrent(), getMean()
- **BoundedRangeStatistic:** getCurrent(), getMean()
-

Listing 12. Different kind of statistics

```
wsadmin> for i in jvmStats.listStatisticNames():
wsadmin>   print i, jvmStats.getJ2EEStatistic( i).getClass()

HeapSizecom.ibm.ws.pmi.stat.BoundedRangeStatisticImpl
UsedMemorycom.ibm.ws.pmi.stat.CountStatisticImpl
UpTimecom.ibm.ws.pmi.stat.CountStatisticImpl
ProcessCpuUsagecom.ibm.ws.pmi.stat.CountStatisticImpl
```

Get more information
See the article Writing PMI applications using JMX interface for more details.

You can retrieve individual statistics from the stats object using:

- **getJ2EEStatistic():** get one PMI metric.
- **getStatistic():**get one PMI metric.
- **getCurrent(), getMean():** get data for BoundedRangeStatistic metric.
- **getCount():** get data for CountStatistic metric.

Listing 13. Getting one statistic from the stats object

```
wsadmin> for i in jvmStats.listStatisticNames(): print i, jvmStats.getJ2EEStatistic( i)

HeapSizename=HeapSize, ID=1, description=The total memory (in KBytes) in the Java
virtual machine run time., unit=KILOBYTE, type=BoundedRangeStatistic, lowWaterMark=
51200, highWaterMark=60149, current=51200, integral=2.6248472575011E13, lowerBound=
51200, upperBound=262144
UsedMemoryname=UsedMemory, ID=3, description=The amount of used memory (in KBytes) in
the Java virtual machine run time., unit=KILOBYTE, type=CountStatistic, count=38213
UpTimename=UpTime, ID=4, description=The amount of time (in seconds) that the Java
virtual machine has been running., unit=SECOND, type=CountStatistic, count=20028681
ProcessCpuUsagename=ProcessCpuUsage, ID=5, description=The CPU Usage (in percent) of the
Java virtual machine., unit=N/A, type=CountStatistic, count=0

wsadmin> jvmStats.getJ2EEStatistic('HeapSize')
name=HeapSize, ID=1, description=The total memory (in KBytes) in the Java virtual
machine run time., unit=KILOBYTE, type=BoundedRangeStatistic, lowWaterMark=51200,
highWaterMark=60149, current=60149, integral=0.0, lowerBound=51200, upperBound=262144

wsadmin> jvmStats.getJ2EEStatistic('HeapSize').getClass()
<jclass com.ibm.ws.pmi.stat.BoundedRangeStatisticImpl at 250875636>

wsadmin> for i in jvmStats.getJ2EEStatistic( 'HeapSize').getClass().getMethods(): print i
public boolean java.lang.Object.equals(java.lang.Object)
public native int java.lang.Object.hashCode()
public java.lang.String com.ibm.ws.pmi.stat.BoundedRangeStatisticImpl.toString()
public int com.ibm.ws.pmi.stat.BoundedRangeStatisticImpl.getStatisticType()
public java.lang.String com.ibm.ws.pmi.stat.StatisticImpl.getName()
public java.lang.String com.ibm.ws.pmi.stat.StatisticImpl.getUnit()
public java.lang.String com.ibm.ws.pmi.stat.StatisticImpl.getDescription()
…
public java.lang.String com.ibm.ws.pmi.stat.BoundedRangeStatisticImpl.toXML()
…
public long com.ibm.ws.pmi.stat.RangeStatisticImpl.getCurrent()
…
public double com.ibm.ws.pmi.stat.RangeStatisticImpl.getMean()
…

wsadmin> jvmStats.getStatistic( 'HeapSize')
name=HeapSize, ID=1, description=The total memory (in KBytes) in the Java virtual machine
run time., unit=KILOBYTE, type=BoundedRangeStatistic, lowWaterMark=51200, highWaterMark=
60149, current=51200, integral=2.6248472575011E13, lowerBound=51200, upperBound=262144

wsadmin> jvmStats.getStatistic( 'HeapSize').getClass()
<jclass com.ibm.ws.pmi.stat.BoundedRangeStatisticImpl at 165808610>

wsadmin> jvmStats.getStatistic( 'HeapSize').getCurrent()
51200L

wsadmin> jvmStats.getStatistic( 'HeapSize').getMean()
1315.7527162741048

wsadmin> jvmStats.getStatistic( 'UpTime')
name=UpTime, ID=4, description=The amount of time (in seconds) that the Java virtual
machine has been running., unit=SECOND, type=CountStatistic, count=19598389

wsadmin>jvmStats.getStatistic( 'UpTime').getClass()
<jclass com.ibm.ws.pmi.stat.CountStatisticImpl at 1288981716>

wsadmin> jvmStats.getStatistic( 'UpTime').getCount()
19598389L
```

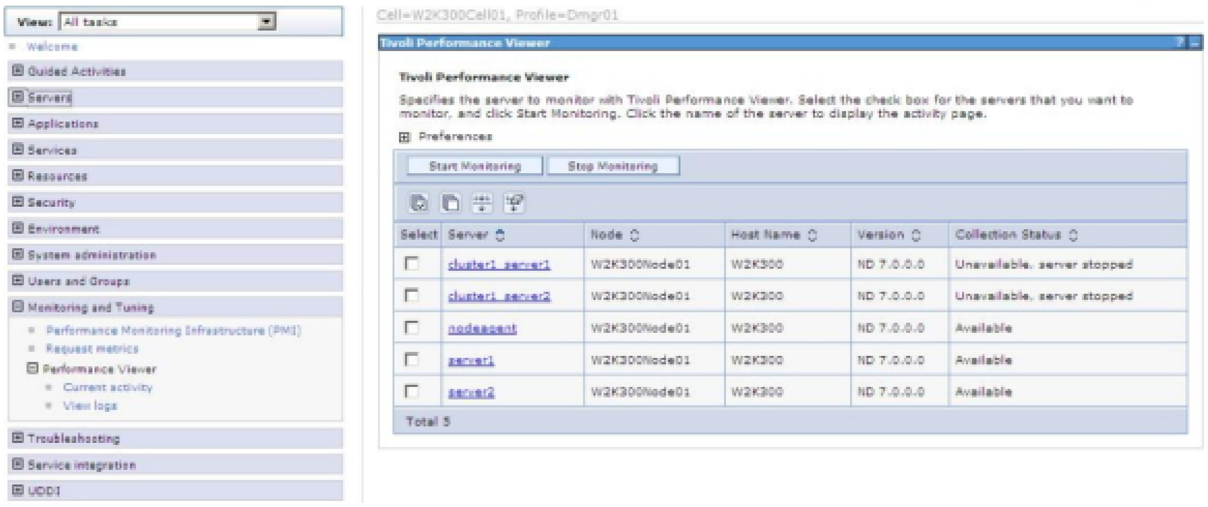
With introspection, you have been able to deep dive into the stats attribute of the JVM MBean and get all the information you need without any ugly string parsing.

Next, discover the simplicity of retrieving PMI data.

Collecting PMI metrics with wsadmin

Using Tivoli Performance Viewer as a guide, collecting PMI metrics with wsadmin and Jython can be very easy. Figure 3 shows a view of available servers using Tivoli Performance Viewer.

Figure 3. Viewing the servers available for PMI (TPV)



Tivoli Performance Viewer

Specifies the server to monitor with Tivoli Performance Viewer. Select the check box for the servers that you want to monitor, and click Start Monitoring. Click the name of the server to display the activity page.

Select	Server	Node	Host Name	Version	Collection Status
☐	cluster1_server1	W2K300Node01	W2K300	ND 7.0.0.0	Unavailable, server stopped
☐	cluster1_server2	W2K300Node01	W2K300	ND 7.0.0.0	Unavailable, server stopped
☐	nodeagent	W2K300Node01	W2K300	ND 7.0.0.0	Available
☐	server1	W2K300Node01	W2K300	ND 7.0.0.0	Available
☐	server2	W2K300Node01	W2K300	ND 7.0.0.0	Available
Total 5					

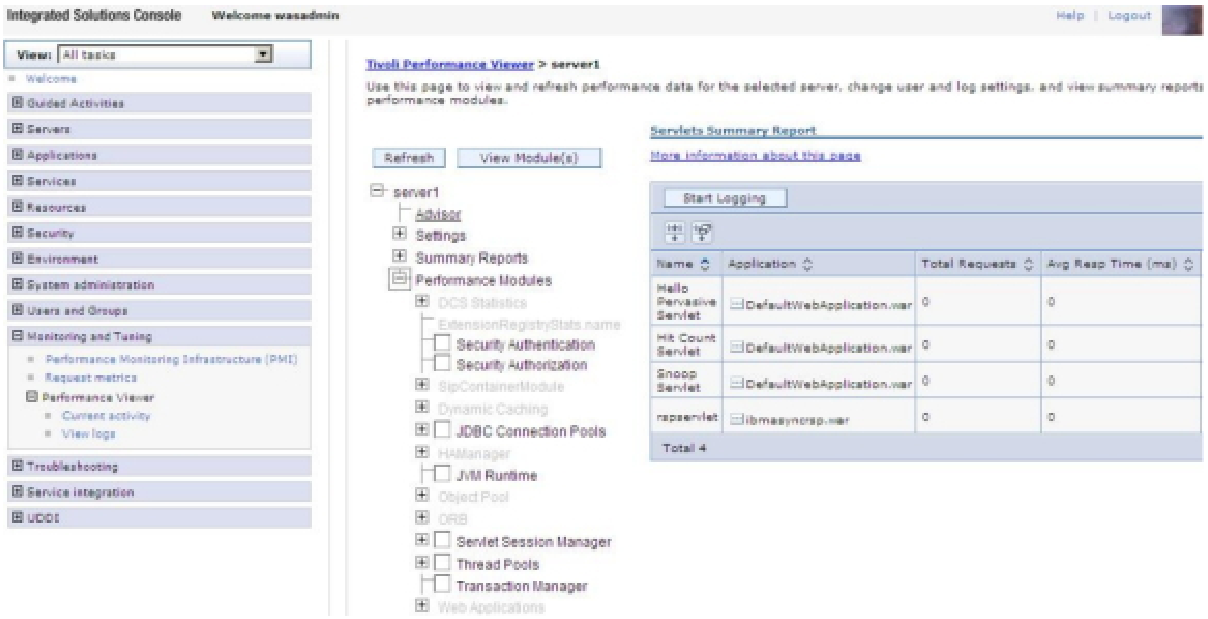
The equivalent view in wsadmin shows that there are three available JMX Mbean of type Perf (Listing 14).

Listing 14. Viewing the servers available for PMI (wsadmin)

```
wsadmin>print AdminControl.queryNames('type=Perf,*')
WebSphere:name=PerfMBean,process=nodeagent,platform=dynamicproxy,node=W2K300Node01,
version=7.0.0.0,type=Perf,mbeanIdentifier=PerfMBean,cell=
W2K300Cell01,spec=1.0
WebSphere:name=PerfMBean,process=server1,platform=dynamicproxy,node=W2K300Node01,
version=7.0.0.0,type=Perf,mbeanIdentifier=PerfMBean,cell=W2
K300Cell01,spec=1.0
WebSphere:name=PerfMBean,process=server2,platform=dynamicproxy,node=W2K300Node01,
version=7.0.0.0,type=Perf,mbeanIdentifier=PerfMBean,cell=W2
K300Cell01,spec=1.0
```

As an example, let's select **server1**. You'll see the top level of the Performance modules hierarchy (Figure 4).

Figure 4. Viewing a server's top level modules (TPV)



Tivoli Performance Viewer > server1

Use this page to view and refresh performance data for the selected server, change user and log settings, and view summary reports performance modules.

Servlets Summary Report

More information about this page

Name	Application	Total Requests	Avg Resp Time (ms)
Hello Pervasive Servlet	DefaultWebApplication.war	0	0
Hit Count Servlet	DefaultWebApplication.war	0	0
Snoop Servlet	DefaultWebApplication.war	0	0
rspservlet	ibmasyncrsp.war	0	0
Total 4			

To get the same view using wsadmin, you need to get the Object Name of server **MBean** and the Object Name of the corresponding **PerfMBean**. Finally, you can invoke the JMX method **getStatsObject()** to retrieve all PMI metrics for this server (with second parameter set to `true`), as shown in Listing 15.

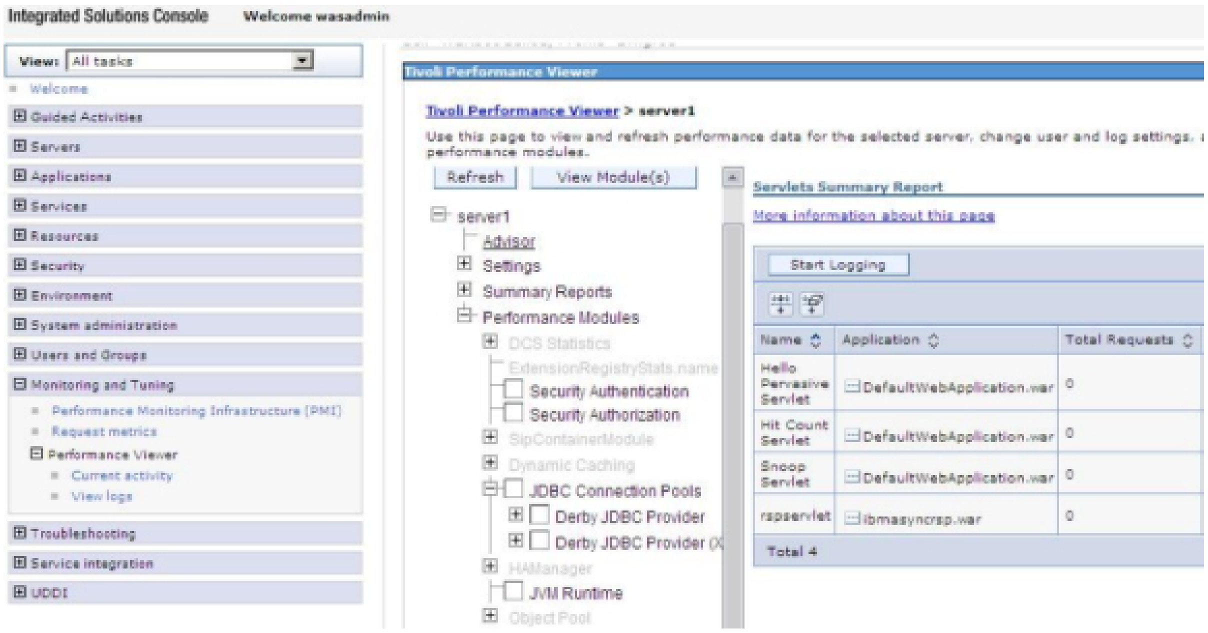
Listing 15. Viewing a server's top level modules (wsadmin)

```
wsadmin> perfStr = AdminControl.queryNames( 'type=Perf,process=server1,
node=W2K300Node01,*')
wsadmin> perfStr
'WebSphere:name=PerfMBean,process=server1,platform=dynamicproxy,node=W2K300Node01,
version=7.0.0.0,type=Perf,mbeanIdentifier=PerfMBean,cell=W
2K300Cell01,spec=1.0'
wsadmin> perfObj = AdminControl.makeObjectName( perfStr)

wsadmin> srvrStr = AdminControl.queryNames( 'type=Server,name=server1, node=
W2K300Node01,*')
wsadmin> srvrStr
'WebSphere:name=server1,process=server1,platform=proxy,node=W2K300Node01,j2eeType=
J2EEServer,version=7.0.0.0,type=Server,mbeanIdentifier=cel
ls/W2K300Cell01/nodes/W2K300Node01/servers/server1/server.xml#Server_1300047119203,
cell=W2K300Cell01,spec=1.0,processType=ManagedProcess'
wsadmin> srvrObj = AdminControl.makeObjectName( srvrStr)

wsadmin> type( perfObj)

wsadmin> stats = AdminControl.invoke_jmx( perfObj, 'getStatsObject', [ srvrObj,
java.lang.Boolean('true')], ['javax.management.ObjectName', 'java.lang.Boolean'])

wsadmin>for i in stats.subCollections(): print i.getName()
wsadmin>
DCSStats.Group
ExtensionRegistryStats.name
Security Authentication
Security Authorization
SipContainerModule
cacheModule
connectionPoolModule
hamanagerModule
```

```
jvmRuntimeModule
objectPoolModule
orbPerfModule
servletSessionsModule
threadPoolModule
transactionModule
webAppModule
wlmModule
```

You can go further and select the **JDBC Connection Pools** module in Tivoli Performance Viewer to see its submodules (Figure 5).

Figure 5. Viewing a server's submodules level (TPV)



Tivoli Performance Viewer > server1

Use this page to view and refresh performance data for the selected server, change user and log settings, performance modules.

server1
- Advisor
- Settings
- Summary Reports
- Performance Modules
  - DCS Statistics
  - ExtensionRegistryStats.name
  - Security Authentication
  - Security Authorization
  - SipContainerModule
  - Dynamic Caching
  - JDBC Connection Pools
    - Derby JDBC Provider
    - Derby JDBC Provider (X
  - HAManager
  - JVM Runtime
  - Object Pool

Servlets Summary Report

More information about this page

Name	Application	Total Requests
Hello Pervasive Servlet	DefaultWebApplication.war	0
Hit Count Servlet	DefaultWebApplication.war	0
Snoop Servlet	DefaultWebApplication.war	0
rspservlet	ibmasyncrsp.war	0
Total 4		

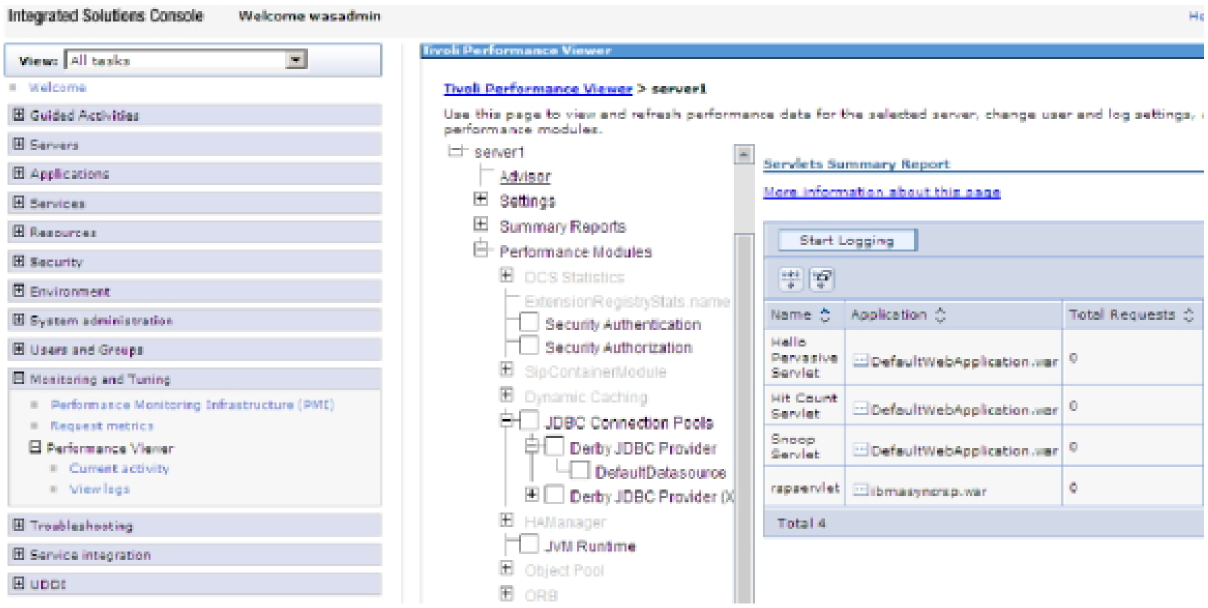
As you have already retrieved all the metrics for the server, to get the same view using wsadmin is easy, as shown in Listing 16.

Listing 16. Viewing a server's submodules level (wsadmin)

```
wsadmin>for i in stats.getStats('connectionPoolModule').subCollections(): print
i.getName()
wsadmin>
Derby JDBC Provider
Derby JDBC Provider (XA)
```

And you can dig one level further, as shown in Figure 6 (Tivoli Performance Viewer) and Listing 17 (wsadmin).

Figure 6. Viewing a server sub modules two level down (TPV)



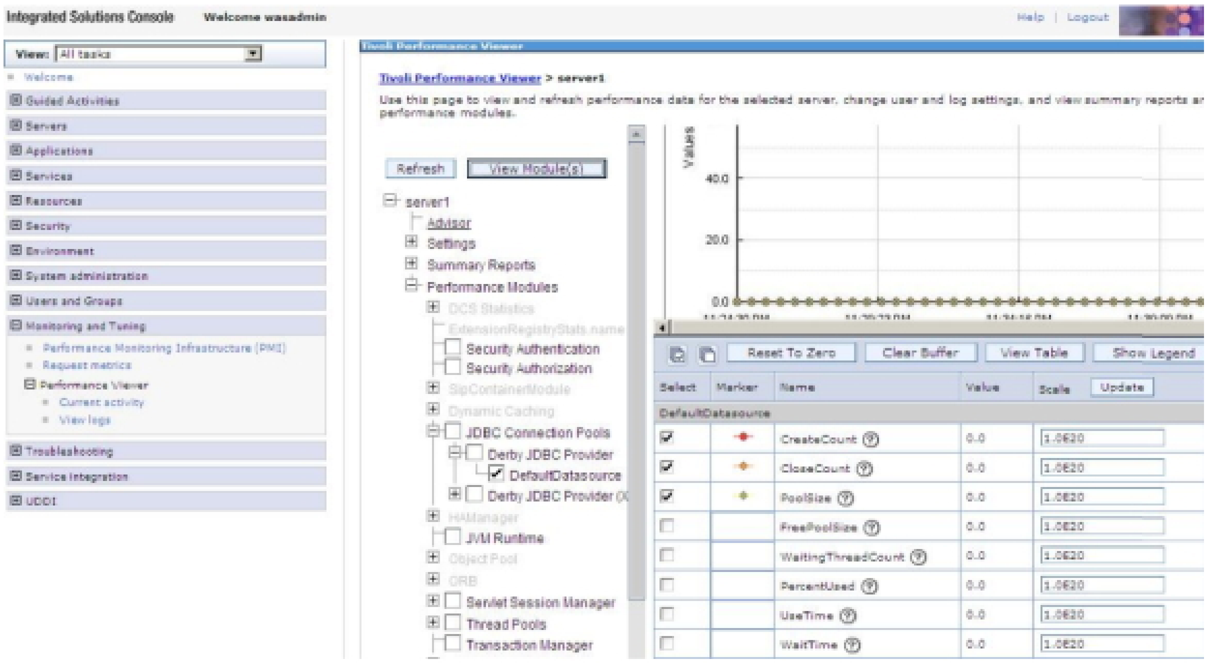
Name	Application	Total Requests
Hello Pervasive Servlet	DefaultWebApplication.war	0
Hit Count Servlet	DefaultWebApplication.war	0
Snoop Servlet	DefaultWebApplication.war	0
rapservlet	ibmasyncrsp.war	0
Total 4		

Listing 17. Viewing a server sub modules two level down (wsadmin)

```
wsadmin> for i in stats.getStats('connectionPoolModule').getStats('Derby JDBC
Provider').subCollections(): print i.getName()
wsadmin>
DefaultDatasource
```

Now, it's time to collect metrics. This is shown first using Tivoli Performance Viewer (Figure 7).

Figure 7. Viewing a server module metrics (TPV)



Select	Marker	Name	Value	Scale
DefaultDatasource				
✓	●	CreateCount	0.0	1.0E20
✓	●	CloseCount	0.0	1.0E20
✓	●	PoolSize	0.0	1.0E20
		FreePoolSize	0.0	1.0E20
		WaitingThreadCount	0.0	1.0E20
		PercentUsed	0.0	1.0E20
		UseTime	0.0	1.0E20
		WaitTime	0.0	1.0E20

The same view using wsadmin can be obtained with the listStatisticNames(), getStatistic(), getCount(), getName() methods (Listing 18).

Listing 18. Viewing a server module metrics (wsadmin)

```
wsadmin> stats.getStats('connectionPoolModule').getStats('Derby JDBC Provider').
getStats('DefaultDatasource').listStatisticNames()
array(['CreateCount', 'CloseCount', 'PoolSize', 'FreePoolSize', 'WaitingThreadCount',
'PercentUsed', 'UseTime', 'WaitTime'], java.lang.String)

wsadmin> stats.getStats('connectionPoolModule').getStats('Derby JDBC Provider').
getStats('DefaultDatasource').getStatistic('CreateCount')
name=CreateCount, ID=1, description=Nombre total de connexions créées., unit=N/A,
type=CountStatistic, count=0

wsadmin> stats.getStats('connectionPoolModule').getStats('Derby JDBC Provider').
getStats('DefaultDatasource').getStatistic('CreateCount').getCount()
0L

wsadmin> stats.getStats('connectionPoolModule').getStats('Derby JDBC Provider').
getStats('DefaultDatasource').getStatistic('CreateCount').getName()
'CreateCount'
```

Simple, isn't it ?

To wrap up our discussion of PMI , let's look finally at PMIService and PMIModule

PMIService and PMIModule

PMIService and **PMIModule** are both configuration entities:

- **PMIService** is a service entity for WebSphere Application Server that is defined in the server.xml config file.
- **PMIModule** is part of PMIService and is the PMI configuration for that server (that is, what you see in the **console > PMI > Configuration tree**).

PMIService is available on each WebSphere Application Server. Its configuration file is **server.xml**, and its attributes include enable, statisticset, and so on. If itPMIService is enabled, the PMI service is started for that application server.

PMIModule's configuration file is **pmi-config.xml**. When PMIService starts, it reads PMIModule information indicated in the pmi-config.xml file.

Listing 19. List of all PMIService

```
wsadmin> print AdminConfig.list( 'PMIService')
(cells/W2K300Cell/nodes/nodedmgr/servers/dmgr|server.xml#PMIService_1)
(cells/W2K300Cell/nodes/node1/servers/server1|server.xml#PMIService_1264758602928)
(cells/W2K300Cell/nodes/node1/servers/nodeagent|server.xml#PMIService_1264758899663)
```

In Listing 20, you can see that the PMIService is not enabled on the dmgr (enable false) and you also have the level of metrics collected (statiscticSet basic).

Listing 20. Status of PMIService for Deployment Manager

```
wsadmin> print AdminConfig.show( '(cells/W2K300Cell/nodes/nodedmgr/servers/dmgr|
server.xml#PMIService_1)')
[context dmgr(cells/W2K300Cell/nodes/nodedmgr/servers/dmgr|server.xml#Server_1)]
[enable false]
[initialSpecLevel []]
[properties []]
[statisticSet basic]
[synchronizedUpdate false]
```

Listing 21. Status of PMIService for Application Server

```
wsadmin> print AdminConfig.show( '(cells/W2K300Cell/nodes/node1/servers/server1|
server.xml#PMIService_1264758602928)')
[context server1(cells/W2K300Cell/nodes/node1/servers/server1|server.xml
#Server_1264758602924)]
[enable true]
[initialSpecLevel []]
[properties []]
[statisticSet basic]
[synchronizedUpdate false]
```

Listing 22. Status of PMIService NodeAgent

```
wsadmin> print AdminConfig.show( '(cells/W2K300Cell/nodes/node1/servers/nodeagent|
server.xml#PMIService_1264758899663)')
[context nodeagent(cells/W2K300Cell/nodes/node1/servers/nodeagent|server.xml
#Server_1264758899663)]
[enable true]
[initialSpecLevel []]
[properties []]
[statisticSet basic]
[synchronizedUpdate false]
```

PMIModule is available on the two servers where the PMIService is enabled (Listing 23).

Listing 23. List of all PMIModule

```
wsadmin> print AdminConfig.list( 'PMIModule')
(cells/W2K300Cell/nodes/node1/servers/server1|pmi-config.xml#PMIModule_1075747243635)
(cells/W2K300Cell/nodes/node1/servers/nodeagent|pmi-config.xml#PMIModule_1)
```

All the attributes of a PMIModule can be displayed using either:

- **AdminConfig.show:** first information level.
- **AdminConfig.showall:** all informations level.

Listing 24. All attributes of a PMIModule

```
wsadmin> pmimodule = '(cells/W2K300Cell/nodes/node1/servers/server1|pmi-config.xml
#PMIModule_1075747243635)'

wsadmin> print AdminConfig.show( pmimodule)
[enable []]
[moduleName pmi]
[pmimodules "[(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|
pmi-config.xml#PMIModule_101)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_102)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_103)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_104)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_105)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_127)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_106)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_111)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_114)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_115)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_116)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_117)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_128)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_129)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_118)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_119)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_120)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_121)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_122)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_123)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_124)
(cells/CEDmgr01/nodes/NDclay111p6Node01/servers/ASNode01-01|pmi-config.xml#PMIModule_126)
]"]
[type []]

wsadmin> print AdminConfig.showall( pmimodule)
[enable []]
[moduleName pmi]
[pmimodules "[[[enable []]
[moduleName alarmManagerModule]
[pmimodules []]
[type alarmManagerModule]] [[enable 35,34,9,1,26,11,12,2,25]
[moduleName beanModule]
[pmimodules []]
[type beanModule#]] [[enable []]
[moduleName cacheModule]
[pmimodules "[[[enable []]
[moduleName *]
[pmimodules "[[[enable []]
[moduleName cacheModule.template]
[pmimodules []]
[type com.ibm.websphere.pmi.xml.cacheModule_template]] [[enable []]
...
```

Conclusion

This article had two main purposes. First, to show that introspection is a real advantage when using wsadmin with Jython. And second, to shake up the Performance and Monitoring Infrastructure to find and easy and efficient way to collect PMI data with wsadmin and Jython.

Acknowledgements

The author would like to extend special thanks Robert (Bob) Gibson and Vishal Charegaonkar for their reviews of this article.

Resources

Learn

- Python Tutorial
- IBM WebSphere scripting with wsadmin: containment paths, configuration IDs and object names
- Writing PMI applications using the JMX interface
- Guide to Python introspection
- Book: WebSphere Application Server Administration Using Jython, Robert A. Gibson, Arthur Kevin, McGrath, Noel J. Bergman, IBM Press, 2010
- How Do You Specify the MBean You Want?
- Class Object: java.lang.Object, java.lang.Class
- Information Center: PMI data organization
- IBM developerWorks WebSphere

Get products and technologies

- Download WebSphere Application Server V8 trial version

About the author

Denis Guillemenot
Denis Guillemenot is a Senior IT Consultant for IBM with 10 years of experience working on monitoring solutions. He is currently working with a client's Level 3 WebSphere Support team.

The Support Authority: Major new usability features in Fix Central make it easy and more convenient to find the fixes you need

Skill Level: Intermediate

Steve Eaton (steaton@us.ibm.com)
IBM Electronic Fix Distribution Architect
IBM

Darrin Lemmer (dlemmer@us.ibm.com)
IBM Fix Central Website Developer
IBM

Todd Mitchell (toddm@us.ibm.com)
IBM Fix Central Website Architect
IBM

07 Dec 2011

Exciting new changes have been made to IBM®'s Electronic Fix Distribution system and its public website, Fix Central. In addition to numerous usability enhancements based on user feedback, a new feature that grabs the set of installed IBM software products on your server and links you to available updates is now being piloted.

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.

What's new in the latest Fix Central release

In November, major enhancements were rolled out to Fix Central, the website that provides software updates to the spectrum of IBM products. These updates are in response to usability research, requests from IBM tech support teams (who provide the software updates you get from Fix Central), and direct feedback from our users.

The most frequent comments we heard were:

- *I want to get to the fixes I need fast, with fewer panels and fewer clicks.*
- *I don't want to have to re-select products, versions and platforms I've already visited before. They should be remembered.*
- *I want it to be easy to search for what I need, and to filter out what I don't.*

Not only have these themes been addressed, but we added one more:

- *I don’t want to track down everything I have that needs and upgrade — I want it figured out for me!*

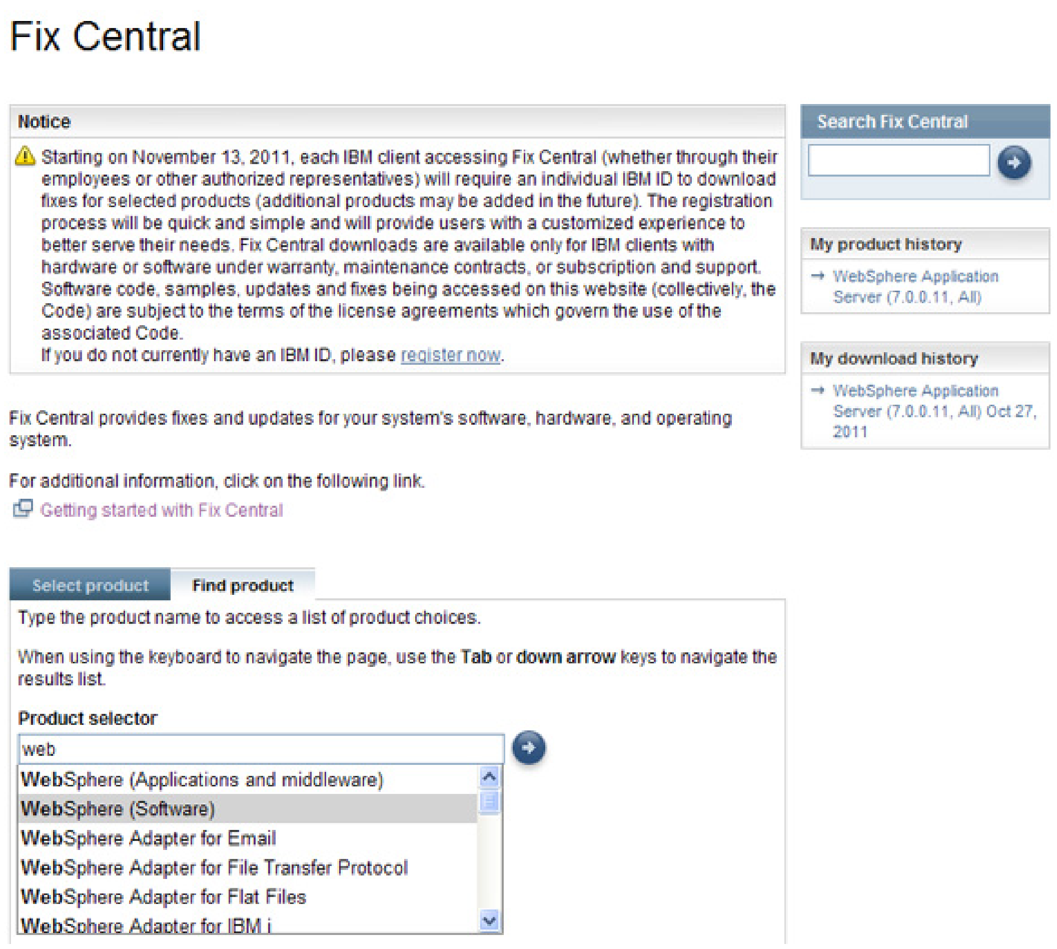
And so we are piloting a major new function, called **inventory upload**, to add even more practicality and convenience to Fix Central.

Below is a high level look at the new search, select fixes, and inventory upload features that now make using Fix Central a whole lot easier.

Search

You'll notice some new features right away on the Fix Central home page, namely the new **Find Product** tab in the center of the page, and **Search Fix Central** control in the upper right column (Figure 1):

- If you know (or have a general idea of) the name of the product you are looking for and don't want to scroll through a long list to find it, select the new **Find Product** tab and simply enter a product name in the **Product selector** field. You can select a product from the list as it displays or continue to type to narrow the list further.
  Figure 1. Fix Central home page



Fix Central

Notice

Starting on November 13, 2011, each IBM client accessing Fix Central (whether through their employees or other authorized representatives) will require an individual IBM ID to download fixes for selected products (additional products may be added in the future). The registration process will be quick and simple and will provide users with a customized experience to better serve their needs. Fix Central downloads are available only for IBM clients with hardware or software under warranty, maintenance contracts, or subscription and support. Software code, samples, updates and fixes being accessed on this website (collectively, the Code) are subject to the terms of the license agreements which govern the use of the associated Code.
If you do not currently have an IBM ID, please register now.

Fix Central provides fixes and updates for your system's software, hardware, and operating system.

For additional information, click on the following link.
Getting started with Fix Central

Search Fix Central

My product history
→ WebSphere Application Server (7.0.0.11, All)

My download history
→ WebSphere Application Server (7.0.0.11, All) Oct 27, 2011

Select product | Find product

Type the product name to access a list of product choices.

When using the keyboard to navigate the page, use the **Tab** or **down arrow** keys to navigate the results list.

Product selector

web

WebSphere (Applications and middleware)
WebSphere (Software)
WebSphere Adapter for Email
WebSphere Adapter for File Transfer Protocol
WebSphere Adapter for Flat Files
WebSphere Adapter for IBM i

- If you already know which fix it is you want, the **Search Fix Central** control will help you find it quickly. Enter keywords about a particular problem, product upgrade level, error message, or a fix ID from IBM support. For example, to find fixes related to server affinity issues in IBM WebSphere Application Server, you could enter `server-affinity WebSphere` and click the arrow button (Figure 2) to get a list of results (Figure 3).

Figure 2. Search Fix Central

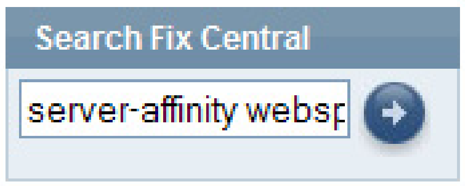
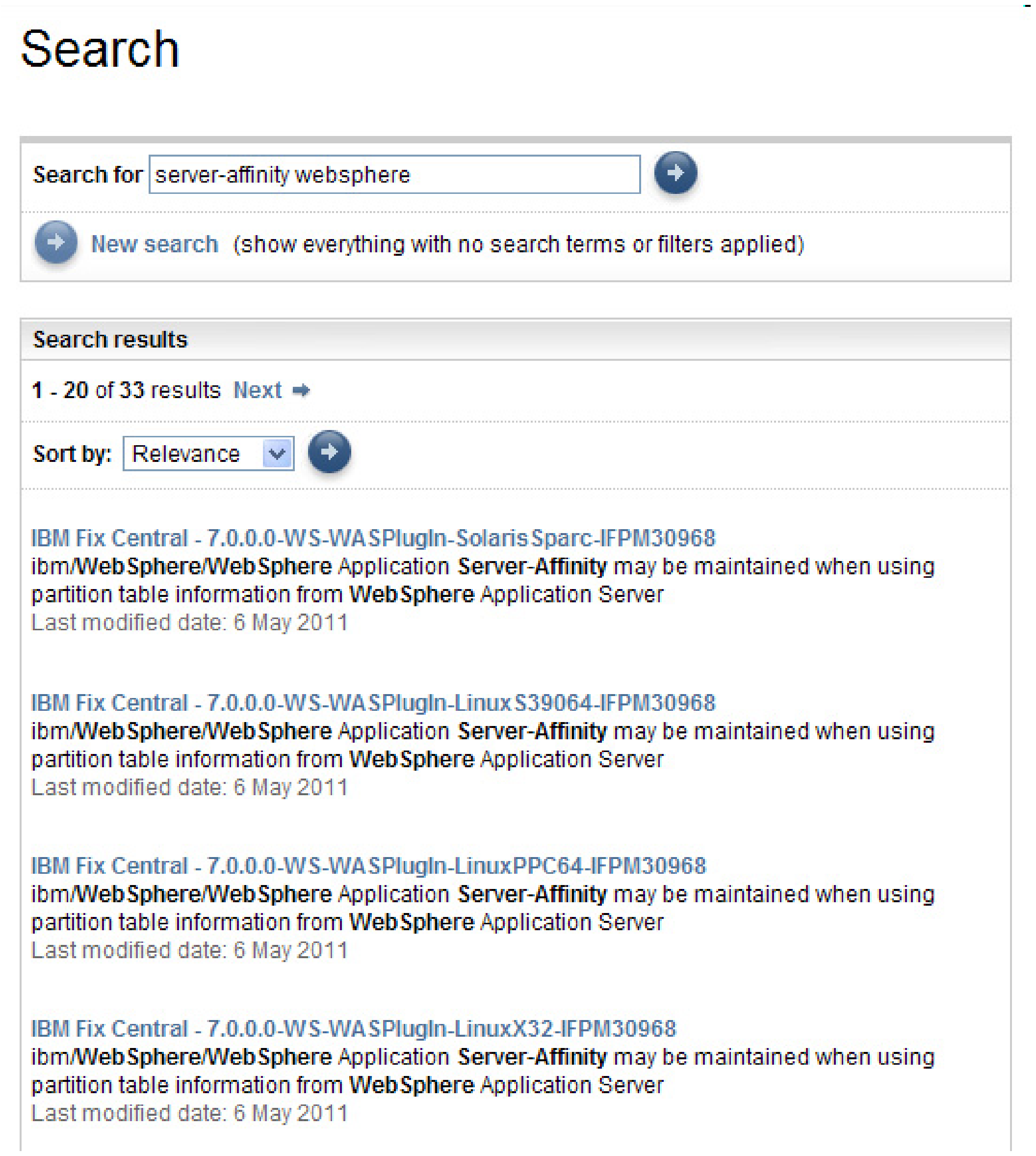


Figure 3. Search results


Search

Search for server-affinity websphere

New search (show everything with no search terms or filters applied)

Search results

1 - 20 of 33 results Next

Sort by: Relevance

IBM Fix Central - 7.0.0.0-WS-WASPlugIn-SolarisSparc-IFPM30968
ibm/**WebSphere**/**WebSphere** Application **Server-Affinity** may be maintained when using partition table information from **WebSphere** Application Server
Last modified date: 6 May 2011

IBM Fix Central - 7.0.0.0-WS-WASPlugIn-LinuxS39064-IFPM30968
ibm/**WebSphere**/**WebSphere** Application **Server-Affinity** may be maintained when using partition table information from **WebSphere** Application Server
Last modified date: 6 May 2011

IBM Fix Central - 7.0.0.0-WS-WASPlugIn-LinuxPPC64-IFPM30968
ibm/**WebSphere**/**WebSphere** Application **Server-Affinity** may be maintained when using partition table information from **WebSphere** Application Server
Last modified date: 6 May 2011

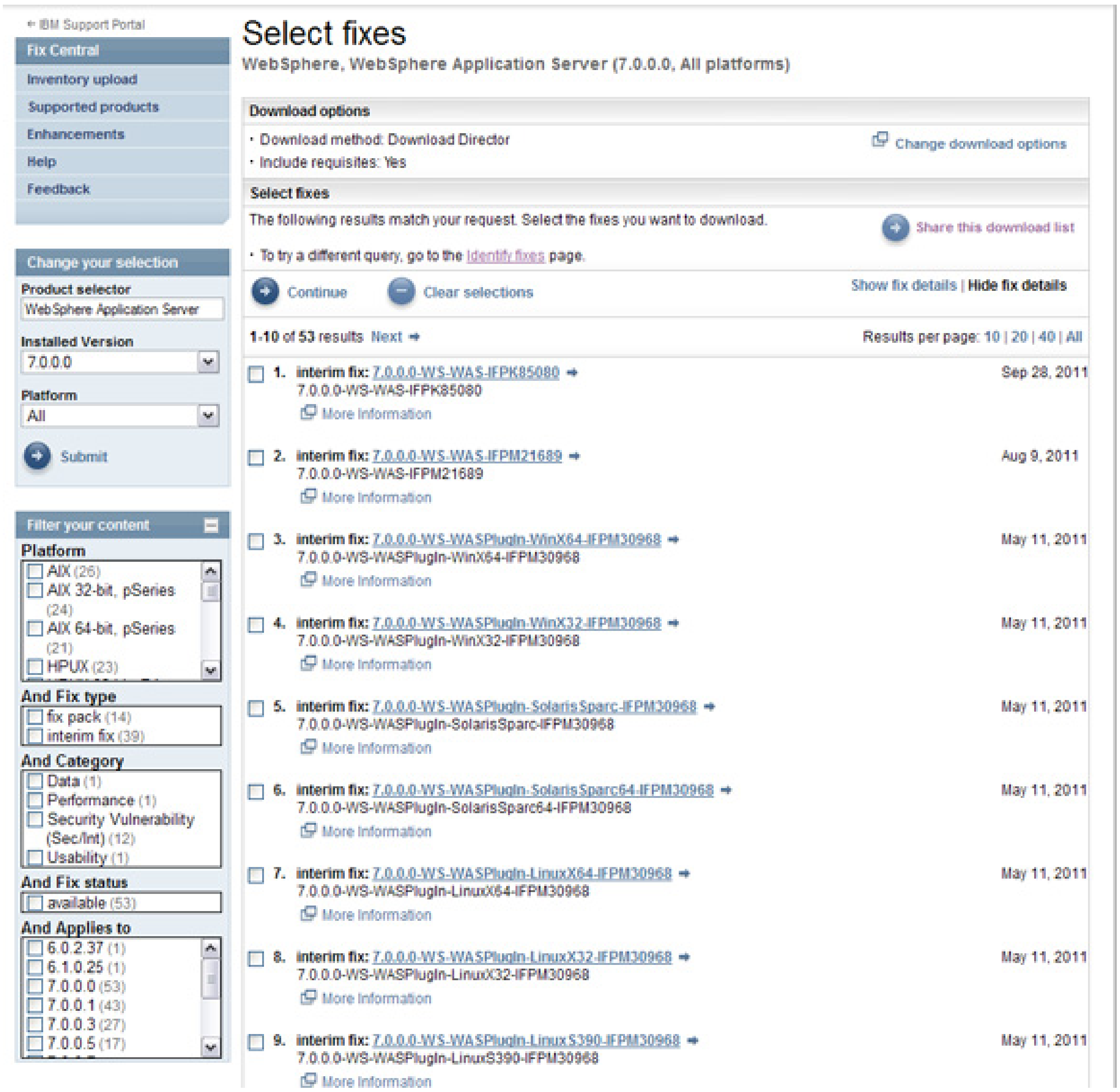
IBM Fix Central - 7.0.0.0-WS-WASPlugIn-LinuxX32-IFPM30968
ibm/**WebSphere**/**WebSphere** Application **Server-Affinity** may be maintained when using partition table information from **WebSphere** Application Server
Last modified date: 6 May 2011

Depending on the keywords you enter and the available content within IBM support, your results could be a mix of Fix Central updates and other related helpful content, such as technical documents.

The **My product history** and **My product downloads** lists in the right column of the home page are new features that relieve you from having to re-select product names, versions, and platforms you already specified in recent visits to Fix Central. My product history pre-fills the product, version and platform fields on the Fix Central home page with values you used earlier (you can modify these values at any time). My download history offers links to product updates that you previously accessed.

Select fixes

Figure 4. Select fixes



Select fixes

WebSphere, WebSphere Application Server (7.0.0.0, All platforms)

Download options

- Download method: Download Director
- Include requisites: Yes

Select fixes

The following results match your request. Select the fixes you want to download.

- To try a different query, go to the Identify fixes page.

1-10 of 53 results Next → Results per page: 10 | 20 | 40 | All

1.	interim fix: 7.0.0.0-WS-WAS-IFPK85080 → 7.0.0.0-WS-WAS-IFPK85080 More information	Sep 28, 2011
2.	interim fix: 7.0.0.0-WS-WAS-IFPM21689 → 7.0.0.0-WS-WAS-IFPM21689 More information	Aug 9, 2011
3.	interim fix: 7.0.0.0-WS-WASPlugIn-WinX64-IFPM30968 → 7.0.0.0-WS-WASPlugIn-WinX64-IFPM30968 More information	May 11, 2011
4.	interim fix: 7.0.0.0-WS-WASPlugIn-WinX32-IFPM30968 → 7.0.0.0-WS-WASPlugIn-WinX32-IFPM30968 More information	May 11, 2011
5.	interim fix: 7.0.0.0-WS-WASPlugIn-SolarisSparc-IFPM30968 → 7.0.0.0-WS-WASPlugIn-SolarisSparc-IFPM30968 More information	May 11, 2011
6.	interim fix: 7.0.0.0-WS-WASPlugIn-SolarisSparc64-IFPM30968 → 7.0.0.0-WS-WASPlugIn-SolarisSparc64-IFPM30968 More information	May 11, 2011
7.	interim fix: 7.0.0.0-WS-WASPlugIn-LinuxX64-IFPM30968 → 7.0.0.0-WS-WASPlugIn-LinuxX64-IFPM30968 More information	May 11, 2011
8.	interim fix: 7.0.0.0-WS-WASPlugIn-LinuxX32-IFPM30968 → 7.0.0.0-WS-WASPlugIn-LinuxX32-IFPM30968 More information	May 11, 2011
9.	interim fix: 7.0.0.0-WS-WASPlugIn-LinuxS390-IFPM30968 → 7.0.0.0-WS-WASPlugIn-LinuxS390-IFPM30968 More information	May 11, 2011

Product selector: WebSphere Application Server

Installed Version: 7.0.0.0

Platform: All

Platform
- AIX (26)
- AIX 32-bit, pSeries (24)
- AIX 64-bit, pSeries (21)
- HPUX (23)

And Fix type
- fix pack (14)
- interim fix (39)

And Category
- Data (1)
- Performance (1)
- Security Vulnerability (Sec/Int) (12)
- Usability (1)

And Fix status
- available (53)

And Applies to
- 6.0.2.37 (1)
- 6.1.0.25 (1)
- 7.0.0.0 (53)
- 7.0.0.1 (43)
- 7.0.0.3 (27)
- 7.0.0.5 (17)

Using the **Change your selection** option, to the left of your search results, you can easily modify your product name, release, or platform search criteria (Figure 4). Below that, the **Filter your content** section is an improvement on the previous filtering function. Find the category you want to filter (Platform, Fix type, and so on) and simply check the items in each category that you want to include. The number of results for each filter is listed beside it. Your search results will refresh automatically with each filter you select or deselect.

With the search filters, It's important to know that:

- Selecting multiple filters within a category will test for an **OR** condition and return all applicable matches. For example, selecting the **AIX** filter with the **AIX 32-bit, pSeries** filter will return all fixes that match **either** of those platform values.

- Selecting filters between categories will test for an **AND** condition. For example, selecting **AIX** under Platform and **7.0.0.0** under And Applies to will return any fixes where the platform value is AIX **and** the applicable version is 7.0.0.0.

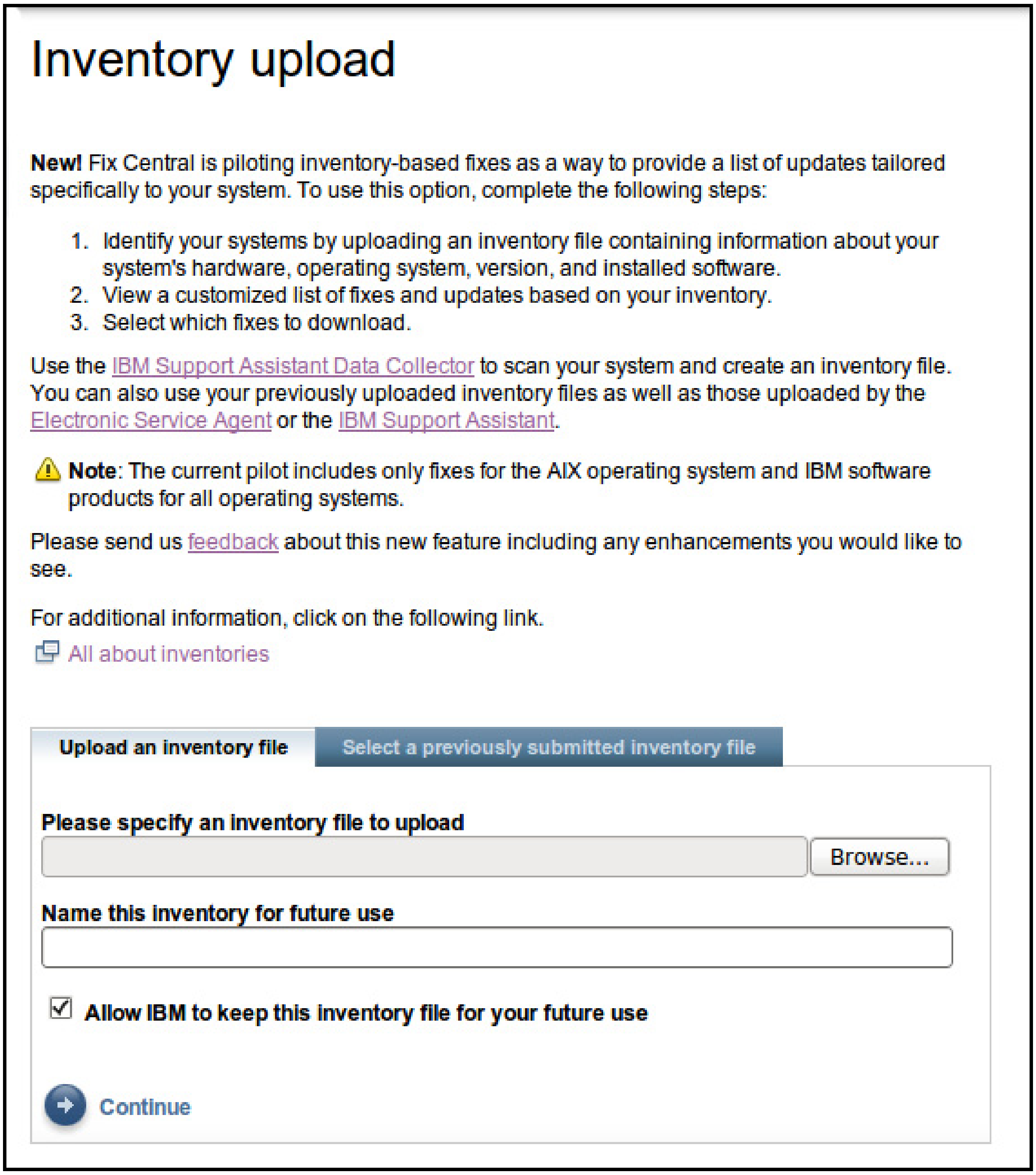
If you have previously used Fix Central, the **Download options** section on the top of the page will display. The download options you used for your last download are remembered and used by default. This provides a reminder of what you previously used and permits you to change the parameters. Download options will not display if no download option preferences have been made.

Inventory upload

New feature
Inventory upload is a new pilot feature and does not yet cover all IBM products. Please use the Fix Central feeback link (Figure 8) to let us know how you like it, or to let us know which IBM products you'd like it to include.

The new **inventory upload** feature (Figure 5) enables you to create a customized list of fixes automatically, based on the IBM software products that are installed on your system. To use this new feature, click the **Inventory upload** link on the left navigation menu from anywhere within Fix Central (Figure 4). From there, you will be able to download the **IBM Support Assistant Data Collector** (a one-time step), which runs on your system and creates an inventory file that can then be uploaded to Fix Central. If you wish, you can name your inventory file and permit IBM to store it for your future use. Previously uploaded inventory files (either through Fix Central, Electronic Service Agent, or IBM Support Assistant) can be selected using the **Select a previously submitted inventory file** tab (Figure 6).

Figure 5. Inventory upload



Inventory upload

New! Fix Central is piloting inventory-based fixes as a way to provide a list of updates tailored specifically to your system. To use this option, complete the following steps:

1. Identify your systems by uploading an inventory file containing information about your system's hardware, operating system, version, and installed software.
2. View a customized list of fixes and updates based on your inventory.
3. Select which fixes to download.

Use the IBM Support Assistant Data Collector to scan your system and create an inventory file. You can also use your previously uploaded inventory files as well as those uploaded by the Electronic Service Agent or the IBM Support Assistant.

Note: The current pilot includes only fixes for the AIX operating system and IBM software products for all operating systems.

Please send us feedback about this new feature including any enhancements you would like to see.

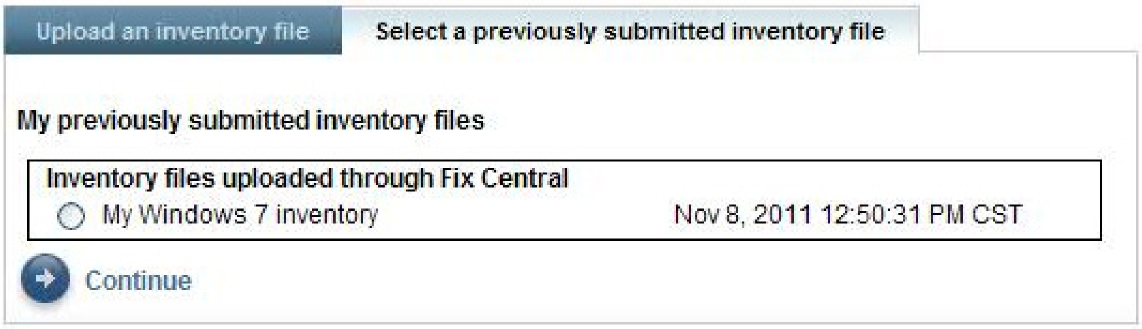
For additional information, click on the following link.
All about inventories

Please specify an inventory file to upload

Name this inventory for future use

☑ **Allow IBM to keep this inventory file for your future use**

Figure 6. Previously submitted inventory files



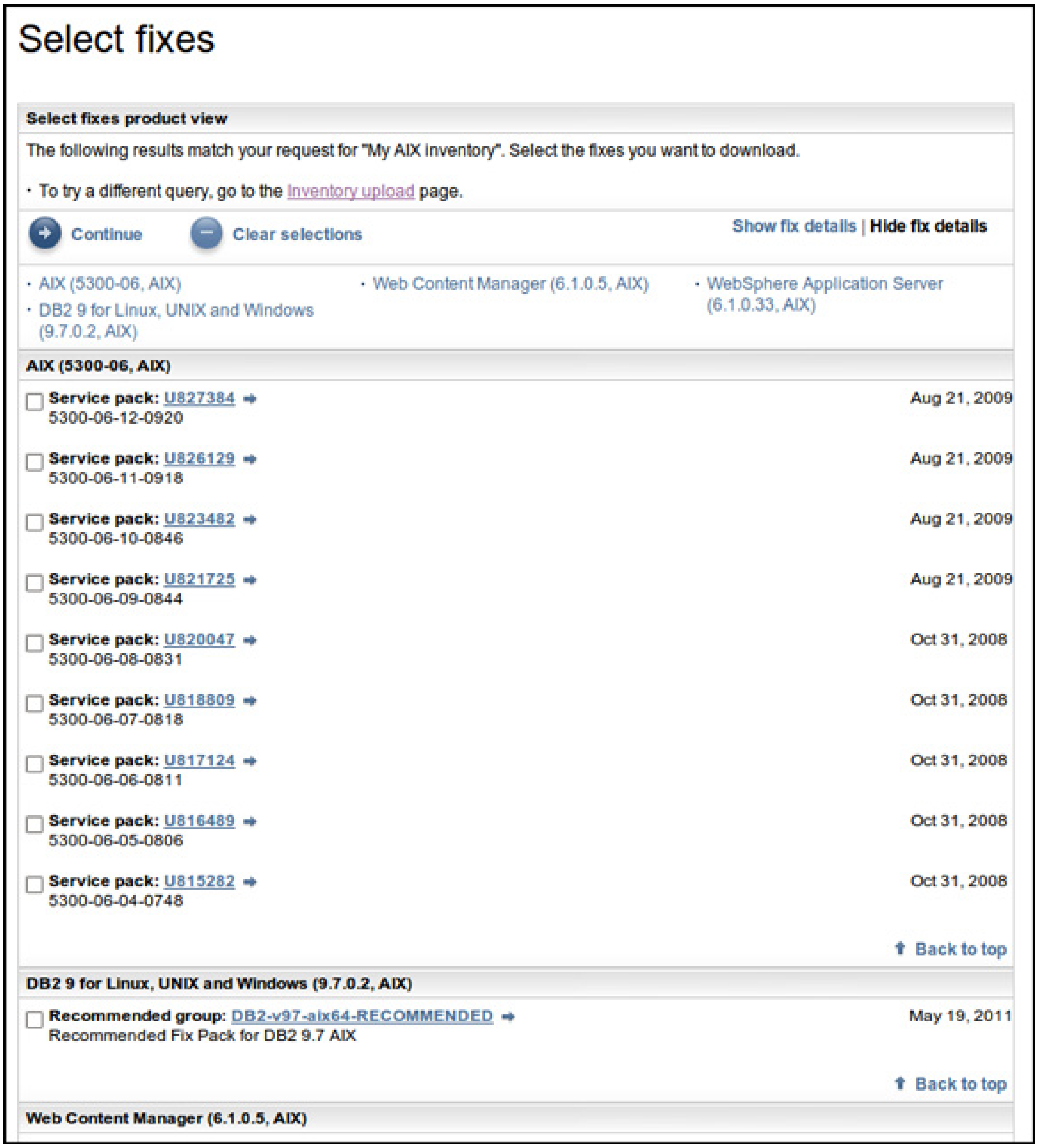
My previously submitted inventory files

Inventory files uploaded through Fix Central

◯ My Windows 7 inventory   Nov 8, 2011 12:50:31 PM CST

Whether you choose to upload a new inventory, or select a previously submitted inventory, clicking Continue on this page will send the inventory file to Fix Central for processing. The Select fixes page (Figure 7) lists the IBM products that were found on your system and lets you select fixes and updates that are available for your products/versions/platforms that are supported by Fix Central. Just select the updates you want to apply and click **Continue** to proceed with your order.

Figure 7. Select fixes for your products



Select fixes

Select fixes product view

The following results match your request for "My AIX inventory". Select the fixes you want to download.

- To try a different query, go to the Inventory upload page.

Continue | Clear selections | Show fix details | **Hide fix details**

- AIX (5300-06, AIX)
- DB2 9 for Linux, UNIX and Windows (9.7.0.2, AIX)
- Web Content Manager (6.1.0.5, AIX)
- WebSphere Application Server (6.1.0.33, AIX)

AIX (5300-06, AIX)

☐ **Service pack:** U827384 5300-06-12-0920	Aug 21, 2009
☐ **Service pack:** U826129 5300-06-11-0918	Aug 21, 2009
☐ **Service pack:** U823482 5300-06-10-0846	Aug 21, 2009
☐ **Service pack:** U821725 5300-06-09-0844	Aug 21, 2009
☐ **Service pack:** U820047 5300-06-08-0831	Oct 31, 2008
☐ **Service pack:** U818809 5300-06-07-0818	Oct 31, 2008
☐ **Service pack:** U817124 5300-06-06-0811	Oct 31, 2008
☐ **Service pack:** U816489 5300-06-05-0806	Oct 31, 2008
☐ **Service pack:** U815282 5300-06-04-0748	Oct 31, 2008

Back to top

DB2 9 for Linux, UNIX and Windows (9.7.0.2, AIX)

☐ **Recommended group:** DB2-v97-aix64-RECOMMENDED Recommended Fix Pack for DB2 9.7 AIX	May 19, 2011

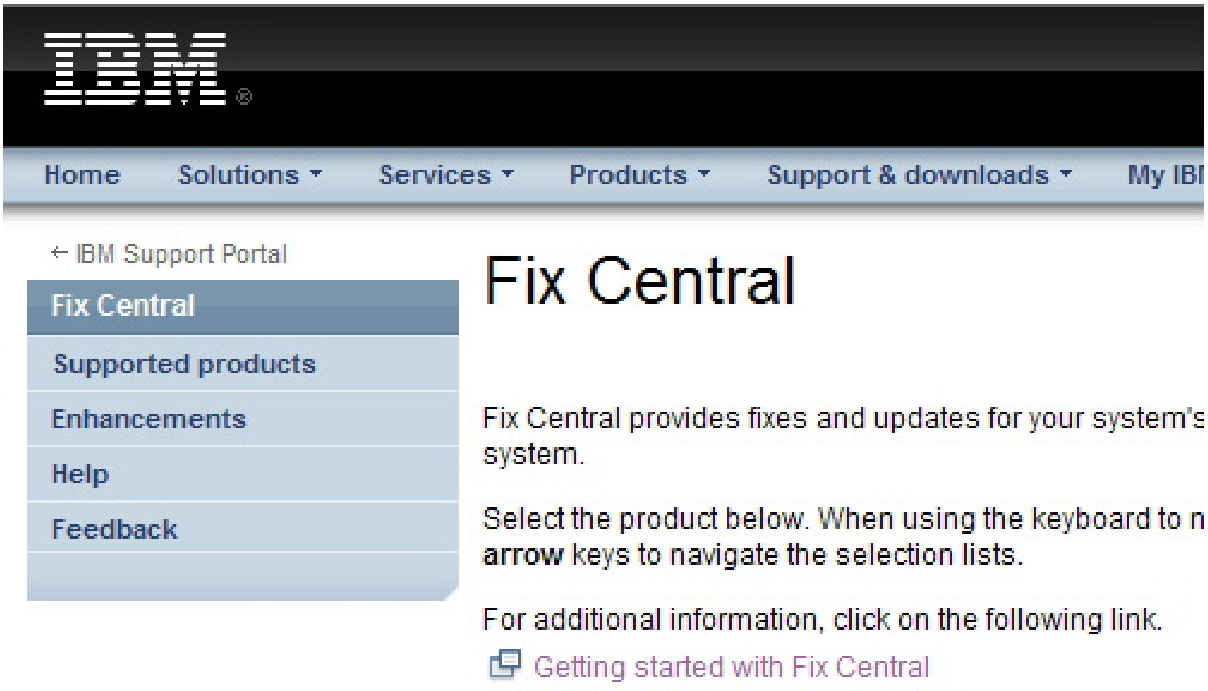
Back to top

Web Content Manager (6.1.0.5, AIX)

Conclusion

We hope you'll like the changes we've made to Fix Central, and, most importantly, that they help make maintaining a system with IBM servers, platforms, and middleware easier and more efficient. But whether you have praise, criticism, questions, or suggestions, please let us know by clicking the **Feedback** link that appears on every page of Fix Central — and please include as much detail as you can. Your input is what makes us better!

Figure 8. Fix Central Feedback



Fix Central

Fix Central provides fixes and updates for your system's
system.

Select the product below. When using the keyboard to n
arrow keys to navigate the selection lists.

For additional information, click on the following link.

Getting started with Fix Central

Acknowledgements

The authors acknowledge John Goodson, Geoff Lubold, Mary Beth Markland, Colin Pfannkuch, Joao Paulo Cordeiro De Souza, Brad Topol, and Karen Wilson, whose combined efforts contributed to this article.

Resources

Learn

- 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

Steve Eaton

Steve Eaton is a Senior Software Engineer based in Austin, Texas, and is the architect for IBM's Electronic Fix Distribution system. He has worked in IBM in support and development roles since 2000.

Darrin Lemmer

Darrin Lemmer is a lead developer with IBM. He has worked at the IBM Rochester, MN lab for 9 years on iSeries Java, iSeries Connect, and the Fix Central website.

Todd Mitchell

Todd Mitchell is an Advisory Software Engineer for IBM, and the Fix Central website architect. He has worked on electronic support websites and other projects at IBM's Rochester, MN lab for the last 20 years.

Comment lines: Extending the benefits of proactive monitoring

Skill Level: Introductory

Alexandre Polozoff (polozoff@us.ibm.com)
Sr. Certified IT Specialist
IBM

07 Dec 2011

Proactive application monitoring helps you detect problems before they have a negative impact. Fine-tuning such activities and understanding where deficiencies exist can improve the health and reliability of the entire processing environment.

Using good results to do something even better

In an early article called Proactive Application Monitoring, I wrote about the benefits of application monitoring and outlined the best metrics to capture in a typical IBM® WebSphere® Application Server environment. The objective of that article was to enable you to detect an anomaly, jiggle a server or application, and ultimately prevent the problem before any user ever got a chance to notice. And that's great. But can we do even better?

Is it possible for operations to reduce the mean time to repair (MTTR) and increase the mean time between failures (MTBR), all before users know anything has happened? If monitoring and operations processes have matured to the point that you can stop problems before they impact users and live operations, there must be more that you can do with this capability to fix more than just immediate problems.

Granularity

All monitors work on some level of **granularity**, as most organizations would be hard pressed to find storage for all the data (much less the horsepower) to analyze and crunch all those numbers. Some monitors collect metrics like CPU and heap size every few seconds. Others, such as those that typically perform polling (like sending an HTTP request to pull a specific page), might only run every 15 minutes.

Naturally, there is a direct correlation between granularity and how quickly a problem is detected. If an HTTP request is sent only once every 15 minutes for Web analytics, then the best you can anticipate is being able to detect a problem every 15 minutes. For a site trying to proactively detect when an error has occurred, this might be too long an interval to wait; probability suggests that users will likely encounter the problem before the monitoring environment does. This leaves little time for troubleshooting and applying a fix before the problem begins to impact the user. A finer granularity of monitoring might be needed in this case. However, running with finer granularity means more activity on the application, which in turn means more data that needs to be collected and stored. Additional storage — although not as expensive as it once was — still needs to be managed, maintained, powered, and backed up.

If the granularity of the monitors is too large and problems are not being detected quickly enough, then you need to come up with a way to accommodate smaller time intervals between tests.

Record keeping

It's a cliche, but history does often repeat itself, and this premise holds true in most application environments. A problem that happens today could be one you will see again in six months or so. A common example I point to is a log disk that gets full (although someone should be monitoring disk space, but that’s for another article). When an application can no longer write to its logs the entire application hangs. If several cluster members are pointed to the same shared log space, then the entire application environment deteriorates rapidly until the application stops. Operations can take a few minutes to troubleshoot, determine what the problem is, and then take steps to correct it.

Problems should be recorded in an operations runbook and include details that describe the problem, what components participated, what people were involved in determining the problem, and what corrective actions were taken. In many cases, it could be appropriate for this documentation to be reviewed by senior management so they can understand how prior business decisions — in this example, the decision to share resources — might have negatively affected the entire application environment and determine if any adjustments should be applied to prevent future occurrences of the same problem. Of course, some problems are the result of not having enough IT staff to participate in troubleshooting and fixing actions.

Access to testing

Having a good test environment is one of the best ways to gain experience working with monitoring tools, granularity, and operational procedures. You get to see firsthand how the tools perform, what impact they have on the applications, and whether the desired granularity is adequate.

Health reporting: Business versus IT

There are at least two factions in any organization that are interested in the health of the processing environment; one side interested in business metrics (how many orders were placed, how many claims were processed, were service level agreements met, and so on), the other in operational metrics (how many component failures occurred, how many applications were restarted, and so on). And the two reports might not necessarily correlate with each other; for example, IT might be concerned because one cluster of Help system applications was down for a considerable part of the day, but the business side is less concerned because the Help system is not considered business-critical.

Operations needs to have a good understanding of business service level agreements and how they differ from operational ones, and therefore know how daily monitoring data should be presented to the different audiences that need this information.

Conclusion

IT operations are under increased pressure to improve efficiencies with less. Going back and reviewing operational readiness and understanding where deficiencies exist will help an organization find better ways of getting monitoring activities tuned to provide an even better and more reliable environment.

Resources

- Proactive Application Monitoring
- IBM's Coremetrics Web Analytics
- Wikipedia: Runbook
- IBM developerWorks WebSphere

About the author

Alexandre Polozoff
Alexandre Polozoff is an IBM Master Inventor and a Senior Certified IT Specialist in the Software Group Performance Technology Practice across the WebSphere product stacks (Application Server, Portal, Process Server, etc) at large topology, international customer sites. He has authored numerous papers and blogs related to performance problem determination around J2EE technologies and speaks at various conferences.

Comment lines: Defeat image sprawl, once and for all

Skill Level: Intermediate

Ruth Willenborg (rewillen@us.ibm.com)
Distinguished Engineer
IBM

07 Dec 2011

Virtualization and cloud computing make it very easy to create new virtual images, but as image catalogs grow, finding and locating the right images gets harder. New images are created because it is easier to create a new image than it is to figure out what existing image might be reusable, creating "image sprawl." Unless you address how to more effectively build and manage your virtual images, you will not realize the full benefits of the cloud. Two new IBM® capabilities, the Virtual Image Library and the Image Construction and Composition Tool, can help you quickly understand the content of your images and build reusable, parameterized images.

Is there an image sprawl monster under your bed?

This May, I switched divisions within IBM® and moved from the WebSphere® organization to Tivoli®. My first week, I met the monster: 11,000 virtual images across just two customers. At that point, I really started looking forward to this month, when I knew we would be releasing two new technologies, the **IBM Image Construction and Composition Tool** and the **IBM Virtual Image Library**. I'd like to introduce you to these new technologies and explain how they can help you control and eliminate the dreaded image sprawl monster.

What causes image sprawl?

Virtual image sprawl is a reasonably new industry phenomena derived from the simplicity that is the "black box" virtual image. Virtualization and cloud computing make it very easy to create new virtual images, but very hard to know what is in the image and how to manage it. Unfortunately, as image catalogs grow, finding and locating the right images gets harder; new images are created because it is easier to create a new image than it is to figure out what existing image might be reusable.

Most customers I see fall into one of two categories. Either:

- They have only standard operating system images, and use scripts and manual installations to add software content to each instance, or:
- They put more content in the images and have an image sprawl problem.

Very few have found an effective balance. The result: virtual image technology is not being leveraged to its fullest benefits.

Right-sizing your image catalog

Perhaps it is because of the time I spent working in performance, but I am a firm believer that to improve the performance of any path, you must **first** look at what you can completely eliminate, and **then** look for ways to make the remaining path faster and simpler. In addition, these concepts also apply to creating a well-balanced image catalog:

- **If you don’t need to do something – don’t do it.**
- **If you only need to do something once, don’t do it every time.**
- **If you can automate it – automate it.**

The advantages of creating a virtual image, by installing and configuring one time and then sharing this image, are tremendous. Only a small number of individuals in your organization need to have installation and configurations skills. The installation and configuration process is executed once and tested. The image is then just copied as needed. This is repeatable and eliminates steps (often manual, error-prone steps), thereby improving quality. The performance advantages are also significant; copying and instantiating an image rather than rerunning installation programs alone is faster, and when combined with image caching technologies, the savings are quite significant.

These advantages are why I believe virtual image templates with more than just the OS make sense; I like to see image catalogs with content. However, when I see image catalogs with thousands and thousands of images, the advantages I just cited are lost: each unique image requires installation and configuration, caching benefits are lost, maintenance is a nightmare, and sharing is often non-existent. You cannot afford to create new images for every variant. There needs to be a balance that supports image reuse and sharing.

There is no one answer to finding this balance. The more you put into an image, the more images you will have to maintain. However, the less you put into images, the more scripts and manual steps you will have, and the more that can go wrong. My three rules of thumb for finding a balance between what to “burn” into the image template once, versus what to leave for instantiation time, are:

- **If you want it in every instance, burn it into the image.** This applies to the operating system and to your organization standards, such as monitoring agents, required security, and auditing software. All these are great candidates to go directly into the image.
- **If it is big or slow, burn it into the image.** This applies to things like large binaries and long running configurations. For example, middleware products such as application and database servers are great candidates to burn the binaries and some level of configuration directly into the image.
- **If it changes frequently, script it at instantiation time.** This applies to fast running configurations, configuration options such as port number, passwords, and so on. This also applies to frequently changing software content such as OS emergency fixes and applications under development.

Implementing the balance using Image Construction and Composition Tool

IBM follows these rules of thumb when creating our IBM Hypervisor Edition images and IBM Workload Deployer (formerly WebSphere CloudBurst™ Appliance) virtual system patterns. The virtual images pre-install the binaries along with multiple configurations. Each image also exposes a set of common configuration parameters. This enables a single virtual image to be used to support many different environments and applications. The image is just copied (cached) and instantiated with a different set of parameters for the desired personality. Pattern scripts are used for cross-configuration between the instantiated instances, because this is fast and there are many specific pattern configurations. In addition, users also add IBM Workload Deployer script packages for frequently changing content, such as their applications.

You can now apply these same techniques to build your own images, by using the IBM Image Construction and Composition Tool. The design of the image construction tool helps you:

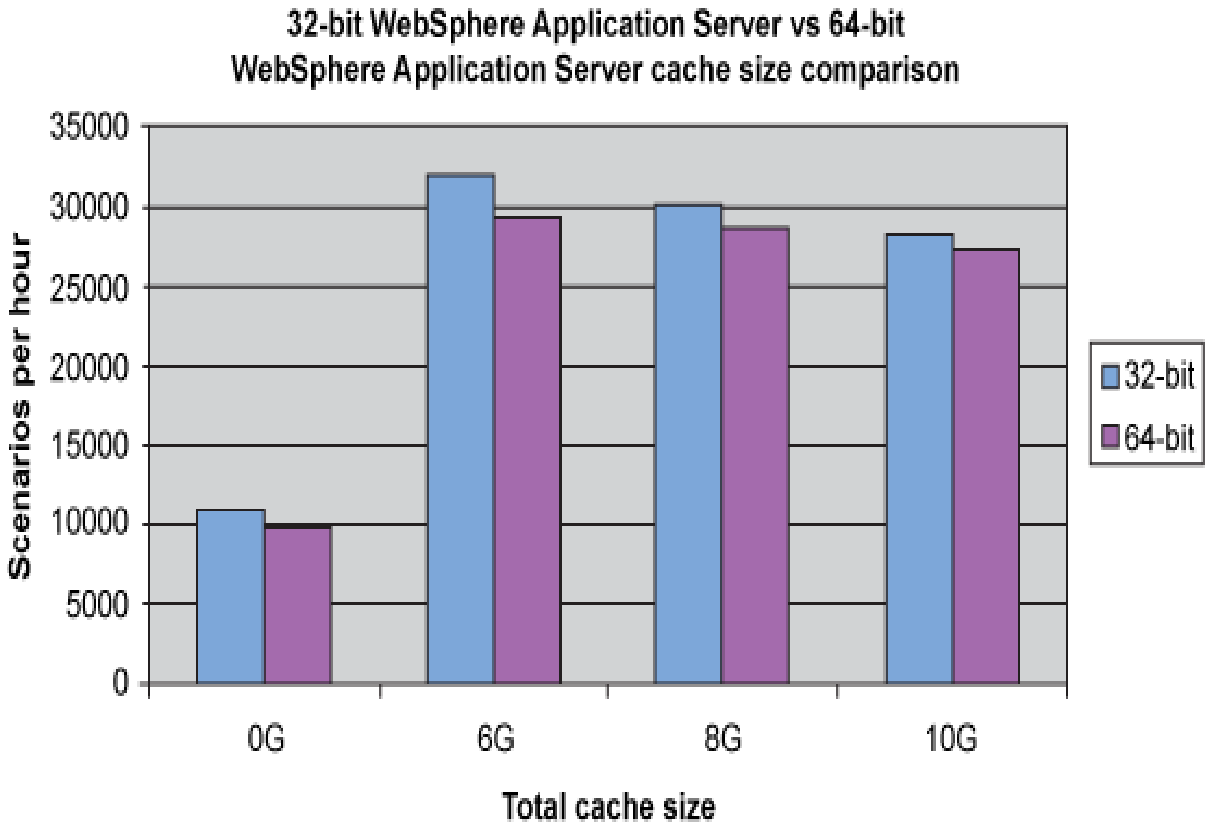
- Automate the one time installation of content into images.
- Create instantiation time configuration options to reduce the number of

images required.

- Recreate an image with a touch of a button.
- Rebuild an image for a different cloud (private <-> public) or on a different OS or version.
- Identify specific software versions and dependencies of an image.

As shown in Figure 1, you can set up one IBM Image Construction and Composition Tool environment and build images for IBM Workload Deployer, IBM SmartCloud Provisioning, IBM SmartCloud Enterprise, or VMware ESX. The bundles and image definitions are reusable across the different cloud environments. The tool ships with IBM Workload Deployer 3.1 and is available as part of the SmartCloud Provisioning 1.2 open beta program.

Figure 1. Overview of IBM Image Construction and Composition Tool

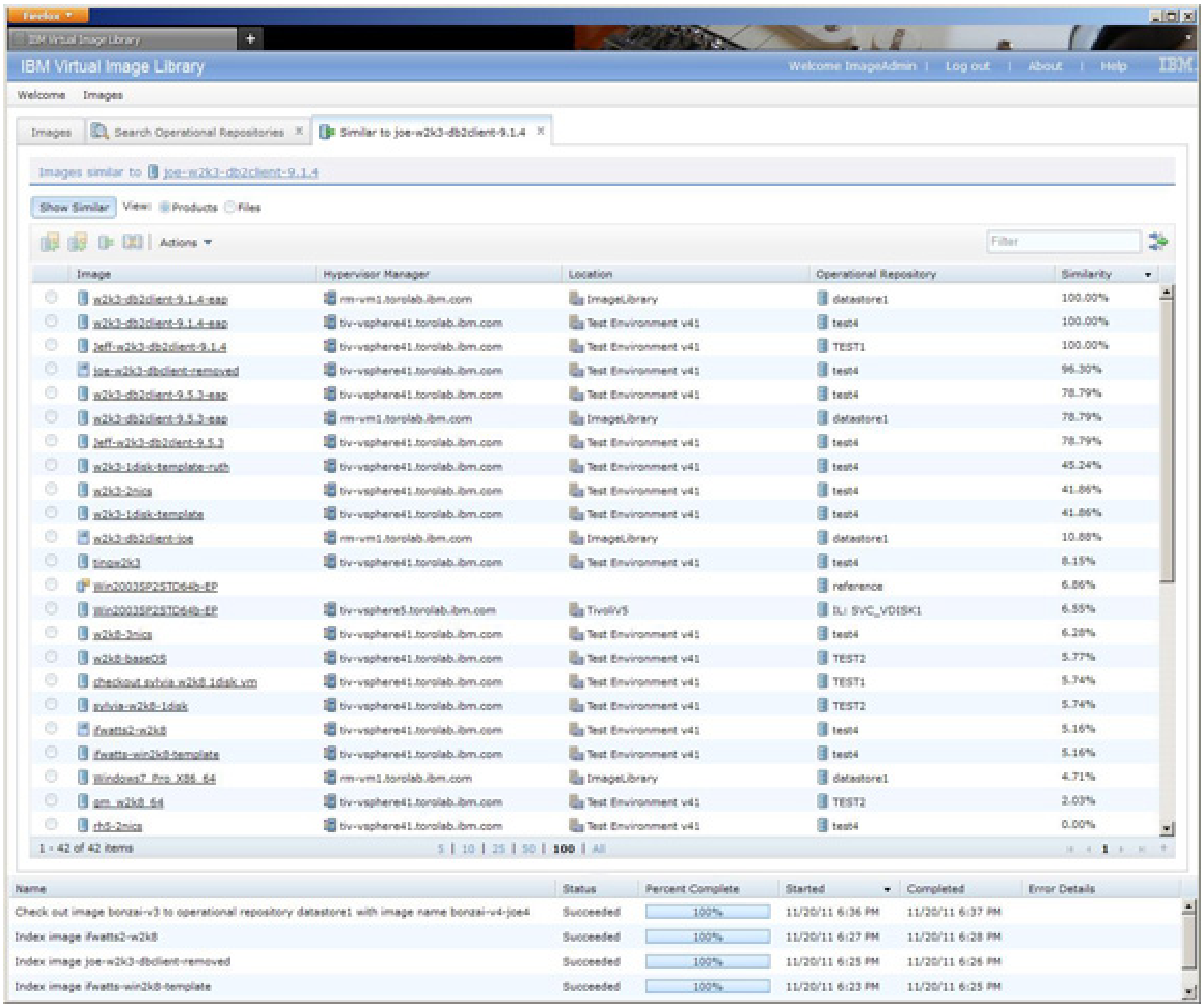


Understanding and controlling your monster with IBM Virtual Image Library

If you already have the image sprawl monster in your environment, the Virtual Image Library capabilities offer a fast and easy approach to help you start eliminating sprawl. First, connect the image library to your existing VMware environments. There is no need to move or copy your images. You can immediately start searching for images with specific software in them, as well as perform similarity and difference reporting. For example, your first step might be to reduce the number of different Windows® or Linux® images that different organizations might have created.

Begin by selecting one of the master operating system images on which you want everyone standardized. Perform a "similarity search" to identify all the similar images. Images that are similar (or identical) to the master image are great candidates for removing. For example, Figure 2 shows an example similarity report, where two images (w2k3-db2client-9.1.4-eap, and jeff-w2k3-db2client-9.1.4) are actually identical to the selected image. You also see images with high similarity. For similar images, you can perform both software and file level differencing comparisons of the similar image against the master to determine exactly what is different. You will likely find that many of the variants are not necessary, and you can start eliminating images.

Figure 2. Example IBM Virtual Image Library Similarity Report


IBM Virtual Image Library
Welcome ImageAdmin | Log out | About | Help
Welcome Images
Images
Search Operational Repositories
Similar to joe-w2k3-db2client-9.1.4
Show Similar
Actions
Filter

Images similar to joe-w2k3-db2client-9.1.4

View: Products Files

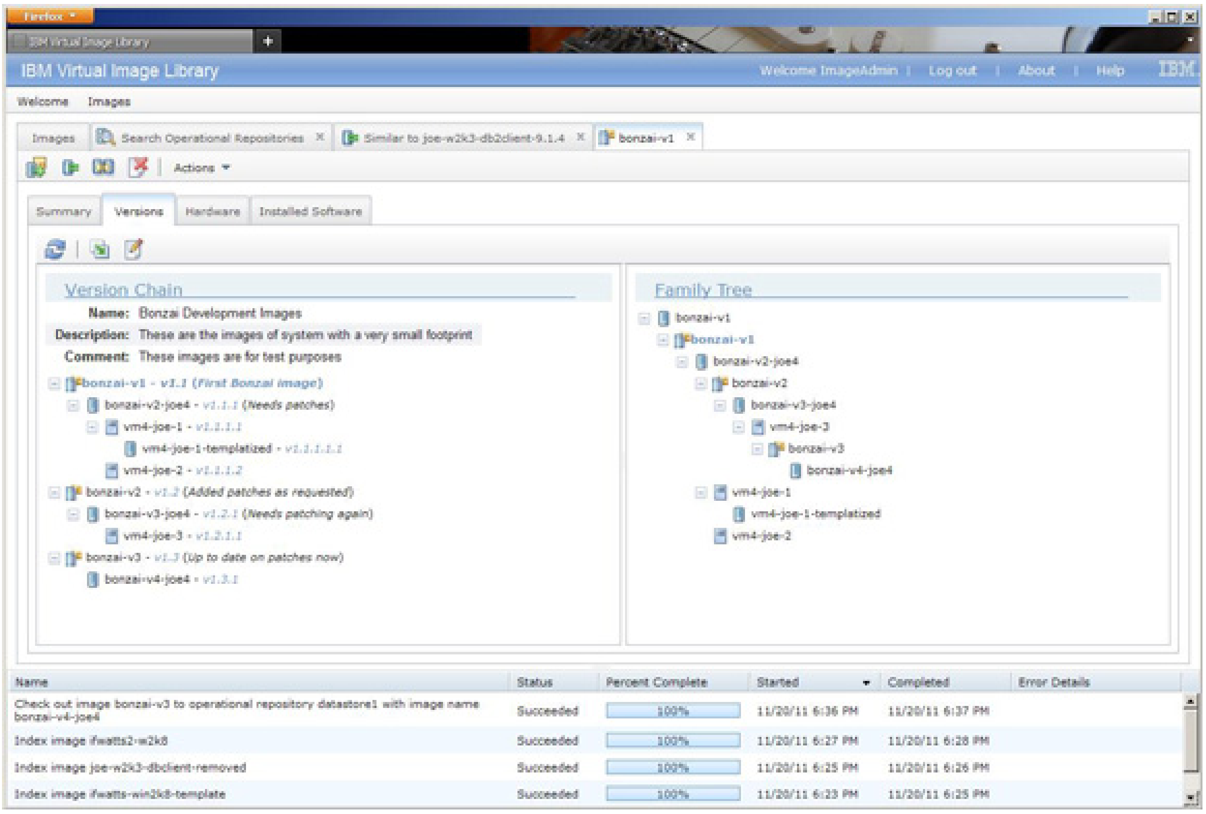
Image	Hypervisor Manager	Location	Operational Repository	Similarity
w2k3-db2client-9.1.4-eap	rm-vm1.torolab.ibm.com	ImageLibrary	datastore1	100.00%
w2k3-db2client-9.1.4-eap	tiv-vsphere41.torolab.ibm.com	Test Environment v41	test4	100.00%
Jeff-w2k3-db2client-9.1.4	tiv-vsphere41.torolab.ibm.com	Test Environment v41	TEST1	100.00%
joe-w2k3-dbclient-removed	tiv-vsphere41.torolab.ibm.com	Test Environment v41	test4	96.30%
w2k3-db2client-9.5.3-eap	tiv-vsphere41.torolab.ibm.com	Test Environment v41	test4	78.79%
w2k3-db2client-9.5.3-eap	rm-vm1.torolab.ibm.com	ImageLibrary	datastore1	78.79%
Jeff-w2k3-db2client-9.5.3	tiv-vsphere41.torolab.ibm.com	Test Environment v41	test4	78.79%
w2k3-1disk-template-ruth	tiv-vsphere41.torolab.ibm.com	Test Environment v41	test4	45.24%
w2k3-2nics	tiv-vsphere41.torolab.ibm.com	Test Environment v41	test4	41.86%
w2k3-1disk-template	tiv-vsphere41.torolab.ibm.com	Test Environment v41	test4	41.86%
w2k3-db2client-joe	rm-vm1.torolab.ibm.com	ImageLibrary	datastore1	10.88%
tingw2k3	tiv-vsphere41.torolab.ibm.com	Test Environment v41	test4	8.15%
Win2003SP2STD64b-EP			reference	6.86%
Win2003SP2STD64b-EP	tiv-vsphere5.torolab.ibm.com	TivoliV5	ILI SVC_VDISK1	6.55%
w2k8-3nics	tiv-vsphere41.torolab.ibm.com	Test Environment v41	test4	6.28%
w2k8-baseOS	tiv-vsphere41.torolab.ibm.com	Test Environment v41	TEST2	5.77%
checkout sylvia w2k8 1disk vm	tiv-vsphere41.torolab.ibm.com	Test Environment v41	TEST1	5.74%
sylvia-w2k8-1disk	tiv-vsphere41.torolab.ibm.com	Test Environment v41	TEST2	5.74%
ifwatts2-w2k8	tiv-vsphere41.torolab.ibm.com	Test Environment v41	test4	5.16%
ifwatts-win2k8-template	tiv-vsphere41.torolab.ibm.com	Test Environment v41	test4	5.16%
Windows7_Pro_X86_64	rm-vm1.torolab.ibm.com	ImageLibrary	datastore1	4.71%
pm_w2k8_64	tiv-vsphere41.torolab.ibm.com	Test Environment v41	TEST2	2.03%
rh5-2nics	tiv-vsphere41.torolab.ibm.com	Test Environment v41	test4	0.00%

1 - 42 of 42 items 5 | 10 | 25 | 50 | 100 | All

Name	Status	Percent Complete	Started	Completed	Error Details
Check out image bonzai-v3 to operational repository datastore1 with image name bonzai-v4-joe4	Succeeded	100%	11/20/11 6:36 PM	11/20/11 6:37 PM	
Index image ifwatts2-w2k8	Succeeded	100%	11/20/11 6:27 PM	11/20/11 6:28 PM	
Index image joe-w2k3-dbclient-removed	Succeeded	100%	11/20/11 6:25 PM	11/20/11 6:26 PM	
Index image ifwatts-win2k8-template	Succeeded	100%	11/20/11 6:23 PM	11/20/11 6:25 PM	

Once you have identified the golden template images, you can use the Virtual Image Library reference repository and version control capabilities to manage them. First, check the images into the Virtual Image Library as Version 1. Then, check the images out to each operational repository so the library can track your golden templates and the locations. When you need to make changes to a golden template, check the image out, apply the changes, and check it in as a new version. Figure3 shows an example of the version tree for an image, as well as the family tree, which shows the explicit lineage.

Figure 3. Example IBM Virtual Image Library Version Chain and Family Tree


IBM Virtual Image Library
Welcome ImageAdmin | Log out | About | Help
Welcome
Images
Images
Search Operational Repositories
Similar to joe-w2k3-db2client-9.1.4
bonzai-v1
Actions
Summary
Versions
Hardware
Installed Software

Version Chain

Name: Bonzai Development Images
Description: These are the images of system with a very small footprint
Comment: These images are for test purposes

```
bonzai-v1 - v1.1 (First Bonzai image)
    bonzai-v2-joe4 - v1.1.1 (Needs patches)
        vm4-joe-1 - v1.1.1.1
            vm4-joe-1-templatized - v1.1.1.1.1
        vm4-joe-2 - v1.1.1.2
bonzai-v2 - v1.2 (Added patches as requested)
    bonzai-v3-joe4 - v1.2.1 (Needs patching again)
        vm4-joe-3 - v1.2.1.1
bonzai-v3 - v1.3 (Up to date on patches now)
    bonzai-v4-joe4 - v1.3.1
```

Family Tree

```
bonzai-v1
    bonzai-v1
        bonzai-v2-joe4
            bonzai-v2
                bonzai-v3-joe4
                    vm4-joe-3
                        bonzai-v3
                            bonzai-v4-joe4
            vm4-joe-1
                vm4-joe-1-templatized
            vm4-joe-2
```

Name	Status	Percent Complete	Started	Completed	Error Details
Check out image bonzai-v3 to operational repository datastore1 with image name bonzai-v4-joe4	Succeeded	100%	11/20/11 6:36 PM	11/20/11 6:37 PM	
Index image ifwatts2-w2k8	Succeeded	100%	11/20/11 6:27 PM	11/20/11 6:28 PM	
Index image joe-w2k3-dbclient-removed	Succeeded	100%	11/20/11 6:25 PM	11/20/11 6:26 PM	
Index image ifwatts-win2k8-template	Succeeded	100%	11/20/11 6:23 PM	11/20/11 6:25 PM	

You can then check this new version out to each location. If possible, remove the previous version from the operational repositories at this time. By doing this, you always have the latest version in use. You can quickly make comparisons between versions to know what has changed, and you can always go back to an older version in the repository, if necessary.

Putting it all together

In addition to using the image library to control the monster, you want to start using IBM Image Construction and Composition Tool to build your image templates. Construct software bundles for content you use across your master images so you can easily automate construction of your master images. Also, start using the instantiation time parameterization capabilities so your images are more reusable, enabling further consolidation of the number of master templates.

To avoid having to change your golden templates for frequently changing content, I recommend that each image template contain a dynamic component to retrieve this frequently changing content on instantiation. There are many techniques for achieving this. IBM has an excellent solution in the IBM Tivoli End Point Manager (previously BigFix®), which even provides pre-built content for operating system patches. IBM Workload Deployer users often use a virtual system pattern script to retrieve frequently changing content (such as applications under development) and install the content at instantiation time.

Conclusion

Virtualization and cloud computing is creating image sprawl problems. Unless you address how to more effectively build and manage your virtual images, you will not realize the full benefits of the cloud. IBM has two new capabilities, the Virtual Image Library and the Image Construction and Composition Tool to help. With Virtual Image Library you can quickly understand the content of your images, search them, and run comparison reports for both differences and similarities. Image Construction and Composition Tool enables you to build reusable, parameterized images.

Check out these capabilities with IBM SmartCloud Provisioning, IBM SmartCloud Enterprise, and IBM Workload Deployer 3.1.

Resources

Learn

- Cloud/Virtualization Management
- IBM Workload Deployer product information
- Establish a system to build custom virtual cloud images: Using the IBM Image Construction and Composition Tool
- IBM developerWorks WebSphere

Get products and technologies

- IBM SmartCloud Provisioning open beta

Discuss

- IBM Image Construction and Composition Tool Forum
- IBM Image Construction and Composition Tool Community

About the author

Ruth Willenborg
Ruth Willenborg is a Distinguished Engineer in IBM Software Group Tivoli. Ruth is currently responsible for Image Management, including image construction, conversion, and management capabilities. Prior to joining Tivoli, Ruth was a founder of the WebSphere CloudBurst Appliance (now called IBM Workload Deployer) and the IBM Hypervisor Edition pre-built virtual images. Ruth has 25 years of experience in software development at IBM. She is co-author of Performance Analysis for Java Web Sites (Addison-Wesley, 2002) and numerous articles on both WebSphere performance and using WebSphere with virtualization technologies.